MASTER EN SCIENCES MATHÉMATIQUES

Mise au point d'essaims de particules proactifs au moyen de la logique floue

Roy, Nicolas

Award date:
2019

Awarding institution:
Universite de Namur

UNIVERSITE DE NAMUR

Faculté des Sciences

MISE AU POINT D'ESSAIMS DE PARTICULES PROACTIFS AU MOYEN DE LA LOGIQUE FLOUE

Mémoire présenté pour l'obtention

du grade académique de master en Sciences Mathématiques

Nicolas ROY

Juin 2019

UNIVERSITE DE NAMUR
FACULTE DES SCIENCES
Enseignement du Département de Mathématiques
Rue de Bruxelles 61 - 5000 NAMUR
Téléphone : + 32(0)81.72.44.90 - Téléfax : + 32(0)81.72.44.64
E-mail : secretariat.math@unamur.be - `http://www.unamur.be`

Mise au point d'essaims de particules proactifs au moyen de la logique floue

ROY Nicolas-Georges

Résumé

Le présent manuscrit décrit l'implémentation et les performances d'un algorithme d'optimisation heuristique adaptatif. La motivation de ce travail était de tirer avantage des énoncés du théorème No Free Lunch dans l'algorithme d'optimisation par essaims particulaires (PSO) au moyen de contrôleurs en logique floue. Pour ce faire, deux types de contrôleurs ont été mis au point afin, d'une part, de contrôler les différents paramètres intervenant dans la PSO classique et dans sa variante quantique (QPSO) et, d'autre part, d'effectuer une hybridation contrôlée entre ces deux variantes. Celles-ci ont été mises au point par le biais d'une méthode robuste, tirant profit du paradigme entités-composants-systèmes, capable de soutenir de futurs développements sur ces contrôleurs. Les variantes implémentées ont été testées sur une gamme de fonctions de test. Le contrôleur flou sur la QPSO a permis de surpasser les différentes variantes de la PSO avec contrôleurs flous de la littérature. L'algorithme hybride flou a quant à lui présenté les meilleurs performances sur l'ensemble des fonctions de test, montrant l'intêret d'un tel contrôle dans une métaheuristique d'optimisation. Au terme de ce travail, nous proposons de poursuivre le développement de tels contrôleurs au moyen de réseaux de neurones optimisés systématiquement.

Mémoire de Master en Sciences Mathématiques

Juin 2019

Promoteur : Timoteo Carletti
Encadrants : Charlotte Beauthier, Anthony Hendrickx, Delphine Nicolay

UNIVERSITE DE NAMUR
FACULTE DES SCIENCES
Enseignement du Département de Mathématiques
Rue de Bruxelles 61 - 5000 NAMUR
Téléphone : + 32(0)81.72.44.90 - Téléfax : + 32(0)81.72.44.64
E-mail : secretariat.math@unamur.be - `http://www.unamur.be`

Proactive particle swarm optimization using fuzzy logic

ROY Nicolas-Georges

Abstract

The present work describes the implementation and efficiency of an adaptive heuristic optimization algorithm. The purpose of this work was to take advantage of the well known No Free Lunch theorem in the particle swarm optimization (PSO) algorithm using fuzzy logic controllers. Two types of controllers have been developed in order to control the parameters involved in classical PSO and its quantum variant (QPSO) and to perform a controlled hybridization between these two variants. These have been developed through a robust method, taking advantage of the entity-component-systems paradigm, able to support future developments on these controllers. All variants were compared on a benchmark of unconstrained test cases. Using fuzzy controller, the QPSO outperformed the different variants of the PSO with fuzzy controllers of the literature. Moreover, the fuzzy hybrid algorithm achieved even better in terms of solution accuracy and robustness over all test cases, unifying the advantages of both PSO and QPSO. Further research will focus on an approach based on neural networks to setup systematically optimized controllers.

Master of Sciences in Mathematics

June 2019

Promotor : Timoteo Carletti
Supervisors : Charlotte Beauthier, Anthony Hendrickx, Delphine Nicolay

Remerciements

Je tiens tout d'abord à remercier Charlotte Beauthier et Anthony Hendrickx pour leur soutien scientifique et humain tout au long de ce mémoire. C'est grâce à nos discussions et au stage que j'ai pu effectuer à Cenaero que ce projet de mémoire a vu le jour. Je remercie également Delphine Nicolay pour le temps consacré à la relecture et ses nombreux conseils ayant définitivement amélioré ce manuscrit. Enfin, je remercie mon promoteur, Timoteo Carletti, pour sa relecture et ses remarques avisées sur mon travail.

Table des matières

Introduction

La recherche d'une solution optimale à un problème est au cœur des préoccupations de bon nombre d'entre nous, parfois sans même que nous en ayons conscience. Par exemple, chercher le trajet le plus court, le plus économe pour se rendre au travail, c'est poser un problème de modélisation et d'optimisation.

Un processus itératif, même s'il est guidé par l'intuition d'un expert, reste une tâche répétitive et fastidieuse. Il n'est ainsi pas concevable que votre GPS attende que vous épuisiez toutes les possibilités de trajet avant de vous indiquer la trajectoire optimale.

C'est pour cette raison que nous avons recours à des méthodes d'optimisation algorithmiques qui peuvent être confiées à une machine. Ces méthodes d'optimisation peuvent se baser sur des expériences mais aussi et surtout sur des simulations, faisant de la modélisation et l'optimisation deux disciplines allant de paire dans la mise au point de solutions aux problèmes d'ingénierie, de recherche ou du quotidien.

Cependant, il arrive que les problèmes posés soient trop complexes, présentant des optima locaux, un profil discontinu ou convexe, le tout couronné d'un nombre de variables imposant. Il est même fréquent qu'aucune formulation analytique du problème ne soit disponible, laissant les méthodes analytiques dans le vide. Pour ces cas, nombres de méthodes dites exactes échouent ou se trouvent extrêmement inefficaces.

Ce défi pousse les chercheurs à mettre au point de nouvelles méthodes d'optimisation afin de surmonter les limitations des méthodes exactes. Il s'agit des *métaheuristiques*, ces algorithmes émulent des phénomènes d'émergence et d'optimisation naturels, par exemple des systèmes biologiques, chimiques ou physiques. La définition d'une de ces méthodes implique cependant un certain nombre de paramètres déterminants pour le bon fonctionnement de l'algorithme. En effet, une métaheuristique est souvent décrite comme une combinaison de deux comportements : l'exploration et l'exploitation [1] dont l'équilibre est régulé par de tels paramètres. L'exploration permet de trouver le véritable optimum en favorisant des solutions nouvelles et originales. L'exploitation quant à elle vise à affiner une solution prometteuse en localisant la recherche dans sa région.

Assurer l'équilibre entre ces deux composantes est une tâche subtile. De plus, le choix de l'algorithme et de ses paramètres dépend largement du problème considéré. C'est pour cette raison que nous allons présenter une voie possible pour surmonter ces deux difficultés avec une méthodologie commune basée sur la logique floue. Dans le cadre de ce travail, nous avons employé deux variantes de l'algorithme *Particle Swarm Optimization* [2] : la PSO quantique [3] et la PSO dite canonique [4]. Notre objectif était de manipuler les paramètres de l'algorithme d'une part et d'autre part d'alterner la méthode employée, soit une hybridation des deux variantes, le tout au moyen de règles de logique explicites. Le contrôle dynamique de paramètres a déjà été décrit dans la littérature mais l'hybridation dynamique floue constitue une implémentation originale pour l'algorithme PSO.

Le présent manuscrit est divisé en trois parties. La première établit le Contexte général de notre recherche, à savoir l'optimisation heuristique et notre collaboration avec CENAERO [1]. A cet effet, nous présentons le concept d'heuristique afin d'introduire le lecteur à l'optimisation émergente, en l'occurrence, l'optimisation par essaims de particules. Nous introduisons également dans cette partie des notions nécessaires de logique floue qui supporte notre implémentation d'essaim *intelligent* ou *proactif*.

La seconde partie développe dans un premier chapitre notre méthodologie concernant l'emploi d'un centre de réflexion au sein de chaque particule au moyen de la logique floue. Nous argumentons nos choix concernant les paramètres à contrôler, les paramètres à mesurer et enfin la logique floue à leur appliquer afin de les lier dans un contrôleur bénéfique pour l'algorithme. Nous abordons aussi dans cette partie l'emploi du paradigme *Entité-Composants-Systèmes* dans la réalisation du code d'optimisation, aux côtés des quelques difficultés rencontrées lors du développement en C++ et Python.

Enfin, la troisième partie est consacrée à l'étude des performances de notre algorithme en comparaison avec l'algorithme génétique de MINAMO, la PSO dite canonique et les variantes de PSO proactives déjà proposées dans la littérature.

1. Centre d'excellence en recherche aeronautique.

Première partie

Contexte

CHAPITRE 1

Motivations et cadre de travail

Ce chapitre est l'occasion de présenter le contexte technologique dans lequel se situe notre travail de recherche avant d'entrer dans le vif du sujet que constitueront les chapitres suivants.

1.1 No free lunch theorem

Ce théorème, formulé par David Wolpert et William Macready [5] rend compte du fait que deux algorithmes d'optimisation, deux heuristiques, seront toujours équivalentes si l'on considère l'intégralité des problèmes possibles. Il s'agit donc d'affirmer que même un algorithme consistant à tirer au hasard des solutions est tout aussi efficace que le plus perfectionné des optimiseurs, à condition de considérer tous les problèmes. L'interprétation de ce théorème est ainsi à prendre avec précautions. En effet, nous ne nous intéressons que rarement à l'ensemble des problèmes. Ce théorème a cependant une autre forme que nous devons aussi à ses auteurs : si un algorithme est performant sur une classe de problèmes, ce sera nécessairement au détriment d'une autre classe. Ce principe est illustré à la Figure 1.1.

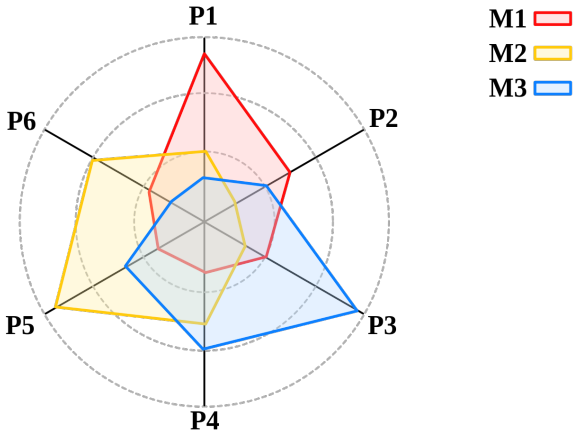


FIGURE 1.1 – **Illustration du No Free Lunch Theorem.** Différentes catégories de problèmes sont réprésentées sur le pourtour du graphe radial, de P1 à P6. L'efficacité de trois heuristiques hypothétiques sont représentées en couleur.

Les motivations de ce mémoire trouvent leur origine dans la volonté d'exploiter cette seconde proposition. Nous souhaitons modifier la méthode de recherche en fonction du problème et de l'état d'avancement de l'algorithme, de manière similaire aux travaux de Shi [6], avec des règles de logique floue. Modifier l'heuristique peut consister à modifier les paramètres de la méthode ou la méthode elle-même, il s'agit là des deux axes de recherche que nous poursuivrons dans ce manuscrit.

1.2 Collaboration avec Cenaero

Notre collaboration a débuté par un stage, effectué durant le second quadrimestre de l'année académique 2017-2018 qui avait pour objet l'intégration d'un algorithme d'optimisation par intelligence d'essaim dans la plateforme d'optimisation et d'exploration de l'espace de recherche Minamo.

Nous avons eu l'occasion d'évaluer différentes variantes de l'algorithme, populaires dans la littérature (PSO11[7], QPSO[3], LDIWPSO[8]) ainsi que d'étudier différentes configurations définissant le comportement de l'algorithme. Les perspectives dégagées suite à ce stage ont conduit directement à cette recherche sur l'hybridation de deux variantes de la PSO, et sur la variation de leur configuration *intelligente* en temps réel.

Ces développements sont ainsi, comme pour le stage, supportés par l'expertise de l'équipe Minamo de Cenaero qui possède une expérience de grande valeur dans le domaine de l'optimisation heuristique.

Fondé en mars 2002 à Gosselies, Cenaero vise à favoriser les innovations technologiques dans les entreprises assurant en interne des activités de conception de produits et/ou de procédés de fabrication. Afin d'aborder les défis de compétitivité et d'innovation posés par les marchés actuels dans divers secteurs d'activité allant des transports au biomédical en passant par l'énergie et le bâtiment. Le développement de méthodes de conception, alliant outils de simulation numérique et algorithmes avancés d'optimisation, a un rôle important à jouer en permettant l'exploration de nouveaux concepts aptes à offrir une réelle innovation. Depuis sa création, le centre a développé des compétences et une expertise reconnue dans le domaine de l'optimisation numérique.

Cenaero possède enfin une infrastructure HPC [1] de pointe sous la forme du supercalculateur *tier-1 Zénobe*. Il s'agit du supercalculateur le plus puissant actuellement en usage en région Wallonne, avec près de 14000 coeurs développant une puissance de calcul de 330 TFLOPS [2]. Cette infrastructure est partagée avec à la fois des partenaires académiques (dont l'UNamur) et industriels de Cenaero.

1. High Performance Computing
2. Milliards d'opérations par seconde.

CHAPITRE 2

Optimisation émergente et intelligence d'essaim

Contents

L'optimisation est une branche des mathématiques visant à résoudre des problèmes de manière analytique ou numérique à travers la minimisation ou la maximisation d'une quantité mesurable, scalaire que nous désignerons le plus souvent par *fonction d'aptitude* dans la suite de ce manuscrit.

Une des premières formes de problème d'optimisation remonte à la géométrie grecque, à travers les travaux en optique de Héron d'Alexandrie au premier siècle de notre ère. Héron a travaillé sur le *principe du plus court chemin*, principe géométrique décrivant la réflexion de la lumière sur la surface d'un miroir. Cependant, il reste hasardeux de dater cette discipline dont les rudiments se retrouvent impliqués dans la création des systèmes fondamentaux à l'évolution des espèces, ainsi qu'à l'organisation des civilisations. Le champ de recherche de l'optimisation cache ainsi, derrière une grande rigueur et un formalisme mathématique important, une part d'inspiration naturelle et de créativité. Cette vision de l'optimisation est particulièrement embrassée par les métaheuristiques qui seront présentées dans ce chapitre, en particulier un algorithme cherchant à émuler l'intelligence d'essaim.

2.1 Métaheuristiques

2.1.1 Formalisation du problème d'optimisation

Avant de continuer ce chapitre, il est nécessaire de poser une définition mathématique de ce que nous entendrons par *problème d'optimisation*. Le problème d'optimisation sur un domaine continu que nous traiterons dans ce travail peut être décrit par le triplet $Q = (S, \Omega, f)$, avec

— S, l'espace de recherche ;
— $\Omega = G_e \cup G_u$, un ensemble de contraintes d'égalité et d'inégalité ;
— f, la *fonction objectif* ou d'*aptitude* à minimiser.

L'espace de recherche S est un sous-espace compact de $\mathbb{R}^n$ où n est la dimension du problème. Le compact est délimité dans $\mathbb{R}^n$ par un hypercube dessiné par le vecteur de position minimale $\mathbf{m}$ et le vecteur position maximale $\mathbf{M}$. L'ensemble de contraintes Ω est un ensemble d'équations et d'inéquations définies sur l'entièreté ou une partie du compact S. Tant que ces (in)équations ne sont pas respectées, la solution est considérée non-faisable et doit donc être invalidée par l'algorithme employé. La solution du problème devra donc, en plus d'appartenir à l'espace de recherche S, faire partie du domaine de faisabilité F.

Enfin, la fonction objectif est une fonction à valeurs dans $\mathbb{R}$

$$\begin{aligned} f \colon S \subseteq \mathbb{R}^n &\to \mathbb{R} \\ \mathbf{x} &\mapsto f(\mathbf{x}). \end{aligned}$$

En nous intéressant à l'optimisation à objectif unique, nous supposons que cette fonction possède au moins un minimum $\bar{\mathbf{x}}$, et des contraintes d'égalité G_e et d'inégalité G_u. Le minimum global s'exprime donc de la manière suivante

$$\exists\ \bar{\mathbf{x}} \in S \subseteq \mathbb{R}^n \colon \forall g_e \in G_e, \forall g_u \in G_u,\ \forall \mathbf{x} \in S$$
$$\begin{cases} f(\bar{\mathbf{x}}) < f(\mathbf{x}) \\ g_e(\bar{\mathbf{x}}) = 0, \\ g_u(\bar{\mathbf{x}}) \leq 0. \end{cases}$$

2.1.2 Ne suffit-il pas d'annuler la dérivée ?

Au 17ème siècle, le calcul différentiel a donné naissance à une nouvelle classe d'algorithmes d'optimisation ramenant le problème d'optimisation à la recherche des racines d'une dérivée. Ce fut le début des très populaires méthodes de descente de gradient, comptant par exemple la méthode bien connue de Newton-Raphson [9]. Ces méthodes sont extrêmement efficaces sur les problèmes unimodaux et continus. Ce calcul de dérivée n'est cependant pas toujours réalisable. D'une part, l'expression analytique de la fonction d'aptitude est rarement connue dans les problèmes réels et d'autre part, elle peut être discontinue ou bien certaines zones peuvent être indéfinies.

Enfin, pour obtenir un algorithme d'optimisation complet, il faut également tenir compte des contraintes. L'approche fondatrice dans ce domaine est celle des multiplicateurs de Lagrange du 18ème siècle. Basée encore une fois sur le calcul différentiel, elle souffre des mêmes limites.

Pour résoudre des problèmes complexes et multimodaux, des méthodes inexactes ont été mises au point. Ces méthodes permettent d'obtenir une solution de qualité dans un temps raisonnable. Elles n'offrent cependant aucune garantie sur l'optimalité de la solution. Ce compromis s'inscrit facilement dans les applications réelles de l'optimisation où une bonne solution faisable en un temps raisonnable est suffisante, faisant des méthodes dites heuristiques une technologie de grand intérêt.

2.1.3 Quelques exemples de métaheuristiques

L'heuristique, du grec ancien *eurisko* signifiant *je trouve* désigne l'art d'inventer et de rechercher. Quand on parle d'une heuristique, on fait référence à un procédé de recherche au sens le plus général.

Ces heuristiques sont inspirées de phénomènes physiques, biologiques ou socio-psychologiques, souvent supportés par le hasard. En effet, les scientifiques se rendent compte dans leur observation de la nature que celle-ci est capable de chercher, de se structurer et d'évoluer à différentes échelles dans une certaine forme d'optimisation continue.

Nous évoquerons par exemple l'organisation des colonies de fourmis autour de leurs sources de nourriture qui a inspiré l'algorithme *Ant Colonies* [10], un algorithme très efficace confronté à des problèmes d'analyse combinatoire tels que le voyageur de commerce [11].

Une autre inspiration de l'optimisation naturelle très populaire est la classe des algorithmes génétiques, exploitant le principe de l'évolution naturelle afin de faire évoluer une population de solutions vers l'optimum. Enfin, l'inspiration peut également venir de phénomènes physiques, comme le recuit simulé [9] ou encore de phénomènes sociaux qui nous concerneront par la suite, en particulier l'intelligence d'essaim qui sera abordée dans la section suivante.

Les heuristiques sont donc diverses et variées et le choix de l'une d'entre elles dépend fortement du problème abordé [1]. Il est même important de modifier les paramètres d'une même heuristique sur différents problèmes comme nous le verrons dans les chapitres qui suivent.

2.1.4 De l'intelligence d'essaim à l'optimisation

L'intelligence d'essaim est une manifestation du concept d'émergence faible, modélisant par exemple les volées d'oiseaux, les colonies de fourmis et les bancs de poissons. Il s'agit de formes et de comportements globaux, qualifiés comme intelligents, émergeant d'un regroupement d'entités simples possédant une portée de communication, de vision et d'action limitée.

Cette intelligence d'essaim a été mise au service de l'optimisation globale de fonctions par James Kennedy et Russell Eberhart [2] sous la forme de l'algorithme PSO. Cet algorithme est directement inspiré de la simulation *Boids* mise au point par Craig Raynolds pour simuler des volées d'oiseaux [12] en 1986. Afin de parvenir à l'émergence d'un comportement d'essaim, ce programme utilise trois règles simples au niveau de ses agents que sont la séparation, la cohésion et l'alignement illustrées à la Figure 2.1.

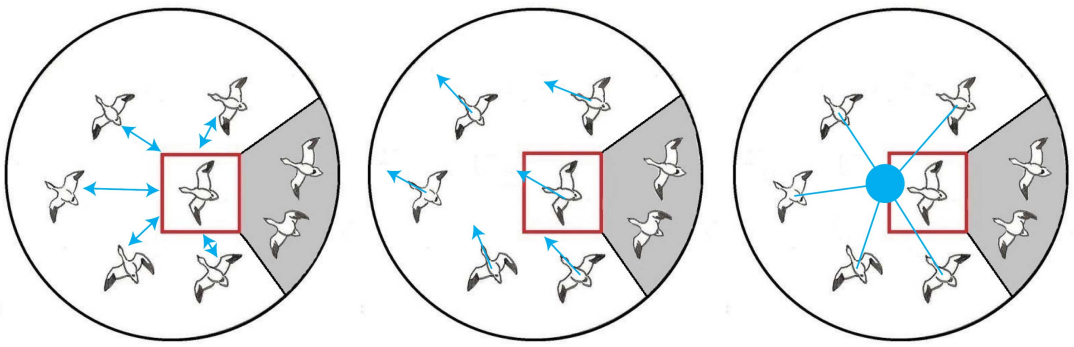


FIGURE 2.1 – **Illustration du code Boïds.** Les trois composantes algorithmiques inclues dans la simulation du vol d'oiseaux sont représentées de gauche à droite : séparation, alignement et cohésion.

La séparation empêche deux oiseaux d'entrer en collision tandis que la cohésion les pousse à se rapprocher. Enfin, l'alignement les conduit à voler dans la même direction. Nous noterons que cette description du programme est inadaptée à l'optimisation car aucun n'objectif n'est poursuivi par un tel essaim. Cependant cet algorithme a connu diverses applications artistiques, notamment dans le film *Batman Returns* pour représenter des volées d'oiseaux.

2.2 Algorithme d'optimisation par essaims de particules

Notre implémentation est basée sur la PSO dite standard, en particulier sa version de 2007 telle que décrite dans l'article de M. Clerc [4], avec certains éléments du standard 2011 [7]. Cette version peut être vue comme la collection des travaux des fondateurs de la PSO renforcée de différentes contributions au niveau de la topologie et de l'initialisation. L'algorithme canonique est représenté à la Figure 2.2. Nous avons de plus apporté nos propres améliorations qui seront décrites en temps voulu.

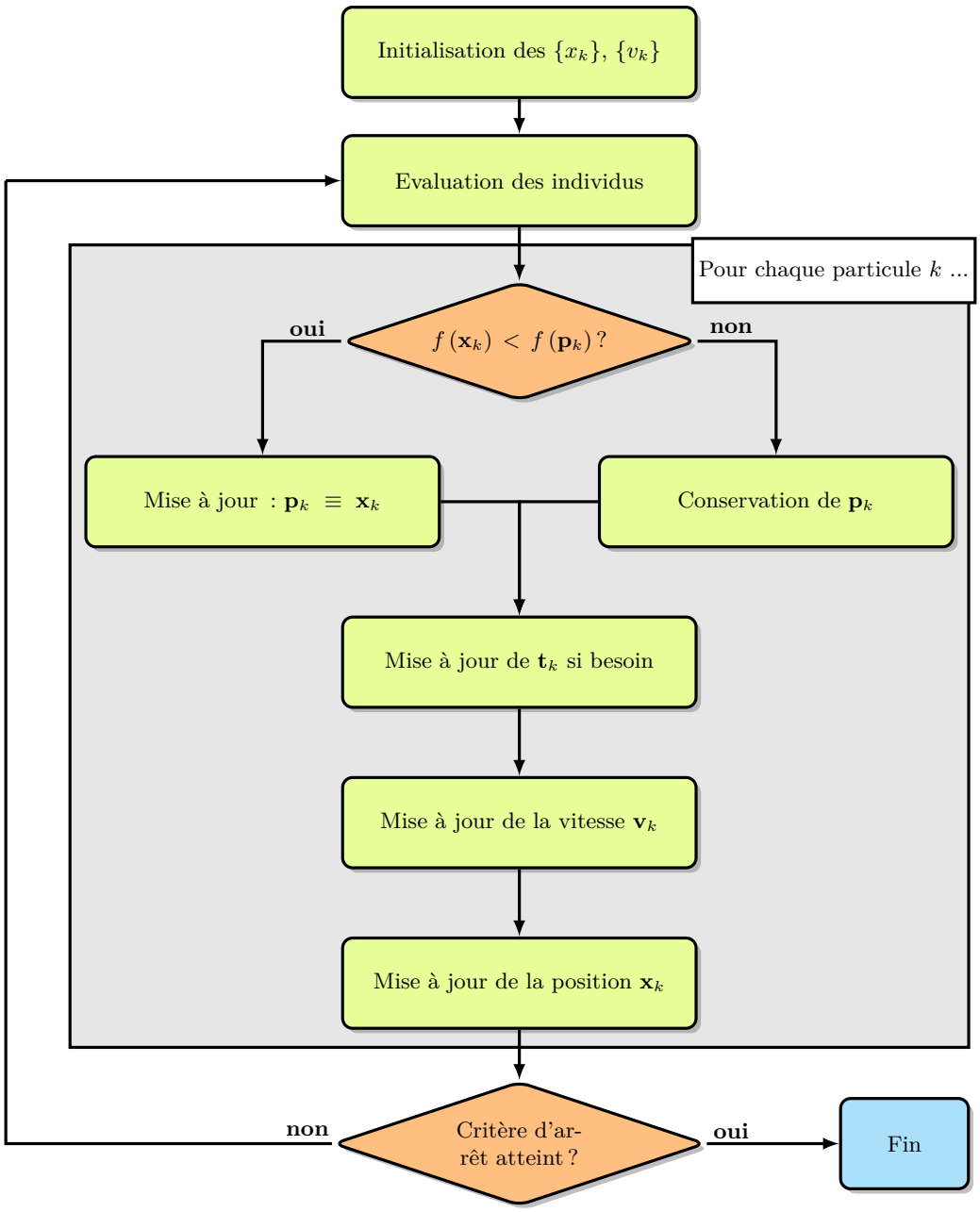


FIGURE 2.2 – **Algorithme PSO canonique.**

2.2.1 Description de l'algorithme

En tant que méthode à population, la PSO emploie un ensemble de particules afin d'explorer l'espace de recherche. Chacune de ces particules possède un comportement lui permettant de réagir à un facteur environnemental alimenté par la fonction objectif ou *aptitude*, faisant ainsi émerger une heuristique au niveau de l'essaim.

Une particule k est définie par une position $\mathbf{x}_k$, définie sur l'espace de recherche du problème d'optimisation et une vitesse $\mathbf{v}_k$ elle-même définie sur un sous-espace S' de $\mathbb{R}^n$. La particule possède également en mémoire sa meilleure position connue notée $\mathbf{p}_k$ ainsi qu'un ensemble d'amis parmi les particules de l'essaim désigné voisinage N_k. Parmi ses amis, une particule définit son meilleur ami, c'est à dire la particule $m \in N_k$ ayant connu le meilleur point dans sa vie $\mathbf{p}_m$, que nous noterons $\mathbf{t}_k$.

Ayant posé les définitions essentielles de la PSO, nous allons par la suite décrire les processus d'initialisation et de mise à jour.

2.2.2 Initialisation de l'essaim

Pour toute méthode à population, l'échantillonnage initial est crucial et conditionne l'intégralité du processus d'optimisation. Un échantillonnage de qualité se doit d'être uniforme et de respecter les bornes afin de représenter au mieux le problème. Nous prenons donc soin d'obtenir un essaim initial couvrant au mieux l'espace de recherche dès les premières itérations.

Nous commençons par générer une population de n_S particules au moyen d'un échantillonnage aléatoire uniforme structuré par un hypercube latin (LHS).

L'utilisation de cette méthode nous offre une bonne répartition des points initiaux dans l'espace de recherche [13]. La méthode d'échantillonnage par hypercube latin respecte la propriété du carré latin de ne posséder qu'un échantillon par colonne et par ligne et ceci en n dimensions. En effet, l'idée derrière cette méthode est de s'assurer que toutes les *portions* de S ont été sondées. Dans le cas où nous souhaitons placer k points dans l'espace, l'algorithme divise l'intervalle associé à chaque dimension indicée i, du sous-espace S, $[m_i, M_i]$ en k intervalles. Les k points sont ensuite placés de telle manière qu'il n'y ait qu'un point dans chaque intervalle de chaque dimension.

Quant à la vitesse des particules, la détermination de la vitesse maximale est importante pour la performance de l'algorithme. Une première approche est de se baser sur la taille du domaine afin de garder ses proportions,

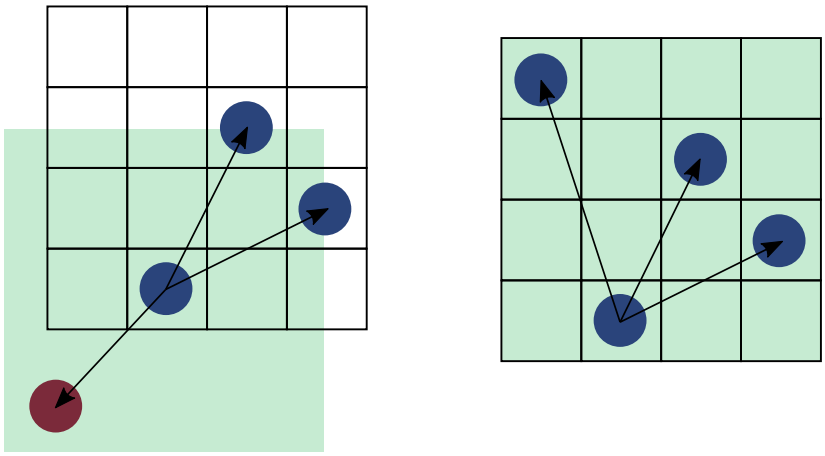
$$\forall k \in [1, n_S],\ \mathbf{v}_k = \frac{\mathbf{U}(\mathbf{m}, \mathbf{M}) - \mathbf{x}_k}{2}.$$

Cette solution conduit cependant à une *explosion* de l'essaim lors des premières itérations, comme illustré à la Figure 2.3a. Particulièrement pour les particules proches des bornes du domaine qui sortent immédiatement du domaine si aucune solution de confinement n'est présente.

Nous avons employé une alternative à cette formule utilisant plutôt la distance aux bornes dont le résultat, représenté à la Figure 2.3b, est donné par

$$\mathbf{v}_k = \mathbf{U}(\mathbf{m} - \mathbf{x}_k, \mathbf{M} - \mathbf{x}_k).$$

Maintenant que nous avons généré un essaim, nous allons aborder dans la section suivante la manière dont la dynamique de recherche est introduite.



(a) Les bornes du domaine multipliées par une constante sont prises comme bornes pour la vitesse.

(b) Le calcul des bornes pour la vitesse tient compte de la position de la particule.

FIGURE 2.3 – **Illustration de deux méthodes pour initialiser la vitesse.** L'étendue dans laquelle le vecteur initial de vitesse peut se trouver est représentée en vert.

2.2.3 Heuristique de l'essaim

Les particules sont animées par une équation de mouvement simple, prenant, à l'itération t pour une particule k, la forme

$$\mathbf{x}_k|_{t+1} = \mathbf{x}_k|_t + \alpha \mathbf{v}_k|_t$$

où α est le taux d'apprentissage de l'essaim.

Afin d'introduire une heuristique permettant de minimiser l'aptitude de chaque particule, la vitesse est mise à jour comme

$$\mathbf{v}_k|_{t+1} = \omega \mathbf{v}_k|_t + r_1 c_1 (\mathbf{x}_k - \mathbf{p}_k) + r_2 c_2 (\mathbf{x}_k - \mathbf{t}_k)$$

où r_1, r_2 sont des variables aléatoires uniformes dans $[0, 1]$. La direction que prend la particule

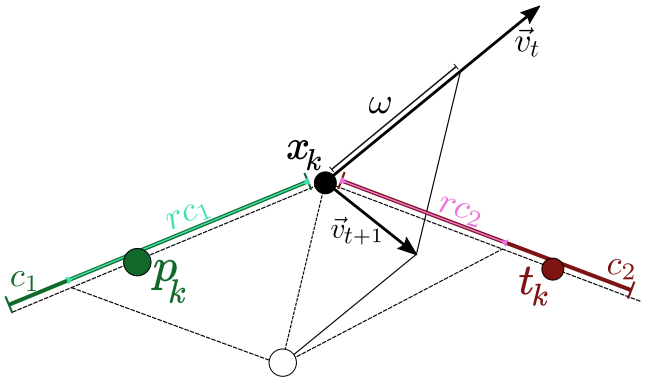


FIGURE 2.4 – **Mise à jour de la vitesse dans l'heuristique classique.** La particule considérée est représentée en noir, son meilleur ami en rouge et sa meilleure mémoire en vert. L'attracteur effectif est représenté en pointillés. Les vecteurs vitesse sont représentés avant (t) et après $(t+1)$ la mise à jour.

est donc dictée par trois termes, représentés à la Figure 2.4. Nous avons d'une part ω, le poids ou l'inertie de la particule qui tend à lui permettre de conserver sa vitesse actuelle. Ce terme

est crucial pour que l'essaim poursuive une exploration satisfaisante du domaine de recherche. Il n'était cependant pas présent dans la formulation de 1995 de la PSO, n'apparaissant qu'en 1998 [14]. Les deux autres termes sont des attracteurs, l'un représente la pression sociale à travers c_2 et le point t_k, le meilleur ami de la particule et l'autre représente la réflexion de la particule à travers un coefficient c_1 modulant l'attracteur vers $\mathbf{p}_k$. Les deux points $\mathbf{p}_k$ et $\mathbf{t}_k$ sont mis à jour au moment de l'évaluation au moyen de la fonction d'aptitude. Le point $\mathbf{p}_k$ de chaque particule k est simplement discrédité si la position actuelle est meilleure, quant au point $\mathbf{t}_k$, meilleure mémoire au sein des amis, il nécessite d'introduire la notion de voisinage, sujet de la section suivante. Dans l'algorithme PSO originel, ce voisinage était l'entièreté de l'essaim, faisant de $\mathbf{t}_k$ la meilleure particule de l'essaim $\mathbf{g}$

2.3 Modifications à l'algorithme

Dans cette section, nous présentons deux modifications de l'algorithme PSO. L'une est une méthode de mise-à-jour alternative qui forme la PSO quantique. La seconde est une réduction de la portée de communication des particules qui donne l'implémentation moderne de la PSO standard.

2.3.1 Une heuristique alternative : l'essaim quantique

Si les essaims de particules canoniques sont inspirés des volées d'oiseaux et des bancs de poisson, l'essaim de particules quantiques est quant à lui inspiré de quelques principes de mécanique quantique.

La mise à jour de la particule ne se fait donc plus au moyen d'une trajectoire, avec une vitesse mais par sauts probabilistes. La position $\mathbf{x}|_{t+1}$ est en effet tirée au hasard autour d'un attracteur situé entre $\mathbf{p}_k$ et $\mathbf{g}$. Nous présentons par la suite la version de l'essaim quantique présentée dans [3]. Cette version se base sur l'observation qu'une particule dans la PSO canonique converge vers un attracteur mobile défini comme le point médian entre $\mathbf{p}$ et $\mathbf{g}$ si les facteurs cognitifs et sociaux sont égaux. Une des conditions pour la convergence de la PSO est donc que toute particule soit liée dans un potentiel autour d'un tel point $\mathbf{a}$.

L'essaim de particules quantiques part de ce constat et pose le problème dans un formalisme quantique, celui d'une particule piégée dans un potentiel. Le profil du potentiel employé détermine la capacité d'exploration de la méthode. En effet, un potentiel de Dirac nous permet d'obtenir un profil de probabilité de présence assez large, comme nous allons le détailler par la suite.

Le potentiel en δ de Dirac représente physiquement une région infinitésimale du domaine dans laquelle notre particule est piégée par un mur énergétique.

Une alternative au potentiel de Dirac est l'oscillateur harmonique. La densité de probabilité aurait alors suivi une distribution plus abrupte permettant moins d'exploration.

Nous commençons par écrire l'Hamiltonien associé à notre particule,

$$\hat{H} = -\frac{\hbar^2}{2m}\Delta^2 + V(\mathbf{x}),$$

dans lequel nous injectons l'expression du potentiel de Dirac centré au point $\mathbf{a}$ tel que

$$\hat{H} = -\frac{\hbar^2}{2m}\Delta^2 - \gamma\delta(\mathbf{x} - \mathbf{a}).$$

Nous en dérivons l'équation de Schrödinger pour une des dimensions

$$\frac{\partial^2\psi}{\partial x_i^2} + \frac{2m}{\hbar^2}\left[E + \gamma\delta(x_i - a_i)\right]\psi = 0.$$

Nous souhaitons cependant connaitre la distribution de probabilité ψ de la particule pour chacune de ces dimensions. Nous savons que la solution est physiquement contrainte par les conditions aux limites

$$\forall i,\ |x_i - a_i| \to \infty \Rightarrow \psi \to 0 \tag{2.1}$$

puisque nous considérons une particule liée au potentiel.

La densité de probabilité, en dehors de la position $\mathbf{x} = \mathbf{a}$ respecte donc

$$\frac{\partial^2 \psi}{\partial x_i^2} = -\beta^2 \psi$$

ce qui nous donne comme solution candidate, en respectant les conditions limites (2.1),

$$\psi(x_i - a_i) = Ce^{-\beta|x_i - a_i|}.$$

En normalisant cette solution, nous obtenons que $|C| = \sqrt{\beta}$, en notant la longueur caractéristique du potentiel $L = \frac{1}{\beta}$, la densité de probabilité prend la forme

$$Q(x_i - a_i) = |\psi|^2 = \frac{1}{L} e^{-2\frac{|x_i - a_i|}{L}}. \tag{2.2}$$

Nous avons désormais à disposition la distribution de probabilité de la prochaine position de notre particule. Cependant, nous avons besoin d'une position classiquement définie afin de procéder à une évaluation d'aptitude et poursuivre la part classique de l'algorithme. Nous allons émuler le phénomène d'effondrement de la fonction d'onde au moyen d'un tirage au sort, imitant ainsi le processus de mesure de la position. Soit une variable aléatoire $u = U_{0,1}$, nous pouvons la substituer dans (2.2) en tant que densité de probabilité Q. Nous obtenons alors la distance à l'attracteur correspondante sur cette dimension en inversant la relation (2.2) de telle manière que

$$x_i = a_i \pm \frac{L}{2} \ln \frac{1}{u}.$$

Cette formulation nous permet de mettre au point une équation de mise à jour pour la position, il nous reste à choisir L qui reste un paramètre libre de notre modèle. Nous posons pour cela

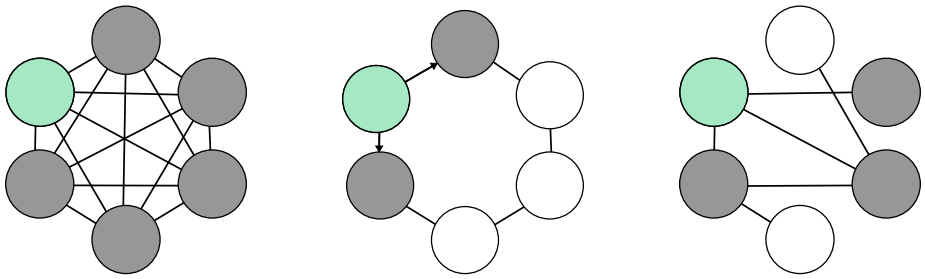
$$L = \frac{|x_i - a_i|}{g},$$

afin de rendre la longueur caractéristique de la distribution de probabilité proportionnelle à l'écart entre la particule et l'attracteur, à travers un unique paramètre g. Si le paramètre g est grand, la particule aura tendance à converger très rapidement vers l'attracteur. Au contraire, si g est proche de zéro, la particule peut apparaitre loin de cet attracteur, favorisant l'exploration.

2.4 Topologie et partage d'information

Nous avons vu dans la section précédente que l'algorithme PSO nous demande de définir le concept de voisinage ainsi que de meilleur ami pour chaque particule.

Afin de définir un voisinage, nous souhaitons naturellement définir une topologie pour notre essaim. Une première solution serait de définir une topologie sur l'espace de recherche, avec une distance donnée par la norme euclidienne. Cependant, une telle notion de distance pour déterminer les particules voisines est numériquement coûteuse, surtout pour des espaces à haute dimension. De plus, elle est compliquée à intégrer dans le cas de variables catégorielles. Une telle vision de la topologie est par exemple employée dans des méthodes telles que le *Brainstorm optimization* [15] au moyen d'une méthode décrivant l'agrégation telle que *k-means*.



(a) Tot. connectée. (b) En anneau. (c) Aléatoire.

FIGURE 2.5 – **Trois graphes topologiques populaires.** La particule partageant l'information est représentée en vert, les particules la recevant sont en gris. Les particules en blanc ne sont pas en communication avec la verte.

L'algorithme PSO emploie quant à lui un nouvel espace afin de définir la topologie, de manière analogue à un réseau social qui ne connaît pas de restrictions géographiques. Différents modèles de graphe topologique sont viables. Le Tableau 2.1 présente les différentes topologies avec une brève description. Le voisinage est le sous-ensemble des nœuds du graphe topologique auquel la particule peut accéder, le nombre de particules dans le voisinage d'une particule étant donné par $\mathcal{K}$. La deuxième grandeur, $\mathcal{C}$, donne une idée de l'intrication du graphe topologique. Il s'agit du nombre moyen de recouvrements entre voisinages de deux particules voisines. La valeur pour la topologie aléatoire a été déterminée pour nos paramètres d'essaim, soit $\mathcal{K} = 3$ et $n_S = 40$, taille de l'essaim.

Nom	Description	$\mathcal{K}$	$\mathcal{C}$
Totalement connectée	L'agent informe tout le monde	n_S	n_S
En anneau	Les maillons de l'anneau communiquent	2	0
Aléatoire	Les informés sont choisis au hasard	3	0.44 [1]
Von Neumann	4 pixels en croix entourent l'agent	4	0
Moore	8 pixels entourent l'agent	8	4

TABLE 2.1 – Quantités associées à différents graphes topologiques couramment utilisés.

Le graphe topologique définit les voies de communication au sein de l'essaim, il vise à limiter l'influence des particules à celles qui leurs sont topologiquement connectées. Différents graphes sont employés en PSO, les plus courants sont représentés à la Figure 2.5.

2.4.1 Topologie aléatoire adaptative

La topologie que nous privilégions dans les essaims de particules est la topologie aléatoire adaptative. Il s'agit d'une topologie où chaque individu désigne aléatoirement un certain nombre de particules avec lesquelles il communique. En cas de stagnation de l'essaim, au niveau par exemple de l'aptitude record, l'assignation des amis de la particule est à nouveau effectuée.

Nous avons sélectionné cette topologie car selon Kennedy [16], une topologie fortement connectée telle qu'une topologie globale conduit à une convergence rapide au risque de tomber plus facilement dans un minimum local. Au contraire, les topologies faiblement connectées ($\mathcal{K}$, $\mathcal{C}$ faible par rapport à S), bien que plus lentes, sont plus performantes sur des problèmes complexes. Ce phénomène est compréhensible étant donné qu'en cas de leader unique pour l'essaim, l'entièreté des particules sera attirée par un seul point, accélérant la convergence mais nuisant à l'exploration.

1. La valeur varie avec la taille de l'essaim et $\mathcal{K}$

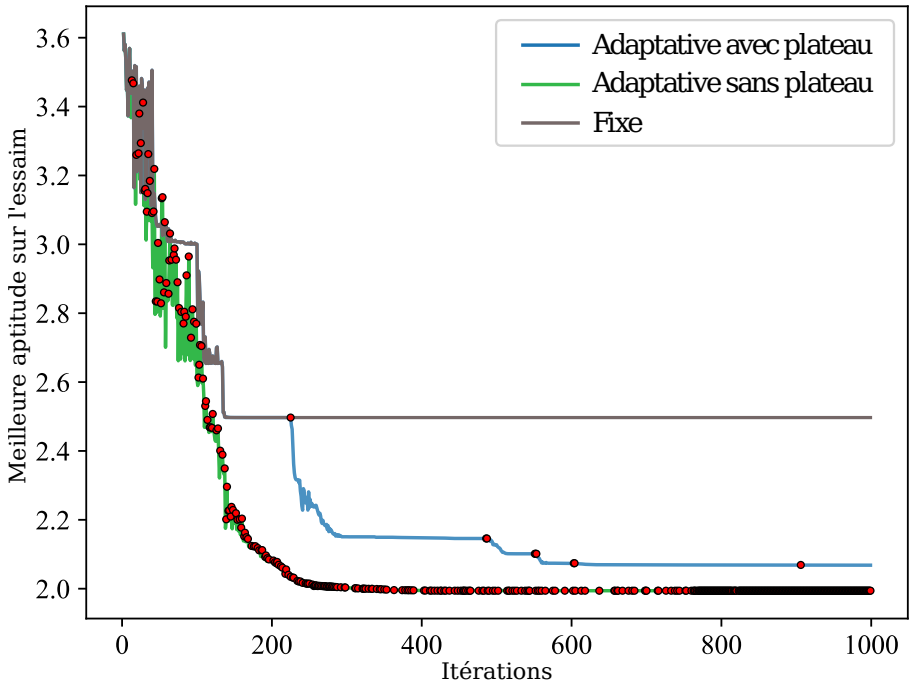


FIGURE 2.6 – **Effet de différentes politiques d'adaptation du graphe topologique.** Les résultats sont obtenus sur une seule graine aléatoire sur la fonction Ackley à dix dimensions afin d'illustrer l'impact des réévaluations indiquées par des disques rouges.

Nous avons confirmé ces observations dans un travail antérieur [17]. De plus, parmi les topologies à faible connexion, nous avons sélectionné une topologie adaptative ; ceci afin d'augmenter la portée de l'information sans pour autant tomber dans les travers d'une topologie forte. En effet, une topologie statique peut condamner certaines particules à travailler en circuit fermé, ce qui peut les conduire dans une situation où elles sont piégées indéfiniment dans un minimum local. De même, dans une topologie qui évite ce premier piège (topologie en anneau), des stagnations arrivent régulièrement car l'information se propage indirectement et l'essaim se stabilise dans des formes contraintes par la topologie. La réévaluation d'une topologie aléatoire ou en anneau est une solution à ce problème. Dès que la performance de l'essaim stagne, le graphe topologique est réévalué, permettant de débloquer la situation en apportant une information nouvelle aux particules.

Dans notre implémentation, la stagnation de l'essaim est décrite par un unique paramètre, σ, le nombre d'itérations stagnantes successives avant réévaluation du graphe topologique. La valeur conseillée pour σ est de 2, soit deux constats de stagnation consécutifs. La stagnation de l'essaim est déclarée quand l'aptitude minimale obtenue sur l'essaim depuis le début du processus d'optimisation n'a pas changé. L'effet de l'adaptation du graphe de partage d'information est représenté à la Figure 2.6, la topologie considérée est aléatoire et le nombre de voisins est de 3. Les réévaluations du graphe topologique sont représentées sur les courbes respectives par des points rouges. On remarque que la topologie aléatoire adaptative conduit à une plus grande qualité de solution que sa contrepartie statique (en gris). Nous notons également qu'une réévaluation moins fréquente, soit seulement si 5 itérations consécutives ne voient pas d'amélioration de l'aptitude minimale est moins efficace (courbe bleue). Il est enfin intéressant de remarquer sur la Figure 2.6 que la réévaluation est clairement à l'origine des sursauts de performance de l'algorithme qui suivent.

CHAPITRE 3

Concepts de logique floue

Contents

Dans ce chapitre, nous présentons la logique floue qui nous a permis d'augmenter la capacité de traitement des agents de la PSO et donc la complexité de leur comportement. Nous commençons par y présenter la logique floue en comparaison avec la logique classique. Les opérateurs et méthodes employées dans notre implémentation sont ensuite précisés.

3.1 Qu'est-ce que la logique floue ?

La logique floue fait partie des logiques alternatives à la logique aristotélicienne bivalente. Une valeur de vérité floue est réelle, généralement comprise dans l'intervalle $[0, 1]$. Cette forme de logique permet de tenir compte de diverses fonctions d'appartenance pour prendre une décision qui s'exprime également comme une valeur réelle. Cette variante de la logique est basée sur le fait que le monde physique, en particulier les humains, ne suivent pas que des règles pouvant être modélisées par une logique bivalente. Il existe toujours une nuance convoyée naturellement par notre langage que la logique floue tente de modéliser au moyen de variables linguistiques et de valeurs de vérité réelles.

Dans la suite, nous présentons les différents aspects de la logique qui doivent être adaptés afin de manipuler des variables floues.

3.2 Exprimer une variable avec nuance

Une première étape incontournable est d'exprimer une valeur, que nous noterons A pour l'exemple, en termes de variables linguistiques. Une variable linguistique est une variable catégorielle prenant pour valeur des propositions. Une variable est donc exprimée dans une base de

propositions au moyen d'un vecteur de valeurs de vérités scalaires.

$$\begin{aligned} |A\rangle &= \sum_{|l\rangle \in \mathcal{L}_A} \langle A|l\rangle \, |l\rangle \\ &= \sum_{i=0}^{n} \rho_i \, |l_i\rangle \, , \end{aligned}$$

avec

$$\mathcal{L}_A = \{|l_i\rangle \, , \; i = 1, n\},$$

l'espace de discours de la variable A qui peut être par exemple, avec $n = 3$: LOW, MEDIUM, HIGH, comme représenté à la Figure 3.2. Nous avons ici emprunté la notation bra-ket propre à la mécanique quantique, un ket représentant un état de variable floue et $\langle a|b\rangle$ représentant la projection de l'état a sur l'état b.

Une fonction d'encodage pour $\mathcal{L}_A$ est une fonction E_A telle que

$$\begin{aligned} E_A : \mathbb{R} &\to [0,1]^n \\ x \mapsto E_A(x) \triangleq |A\rangle &= \begin{bmatrix} M_{l_1}(x, \mathbf{a}) \\ \vdots \\ M_{l_n}(x, \mathbf{a}) \end{bmatrix}, \end{aligned}$$

avec le vecteur constant **a** associé à la variable à encoder. Ce vecteur **a** est un ensemble de paramètres définissant la forme des fonctions d'appartenance $M_i(x, \mathbf{a})$. Ces fonctions d'appartenance sont de simples fonctions de $S_i \subset \mathbb{R}$ dans $[0, 1]$ où S est un compact contenant les valeurs continues prises par la variable à encoder. Des formes typiques pour ces fonctions sont des sigmoïdes, des trapèzes ou encore des gaussiennes.

Exemple

Considérons un **système** de climatisation. La température est qualifiée par une variable linguistique prenant les valeurs "Cold", "Moderate" et "Hot". Un **état de la sonde** de température est ainsi qualifié par un triplet de scalaires représentant la proportion de chaque valeur, qui dans le cas d'une pièce froide prend des valeurs telles que

$$|\text{Temp.}\rangle = \begin{bmatrix} 0.9 \; \textit{Cold} \\ 0.1 \; \textit{Moderate} \\ 0.0 \; \textit{Hot} \end{bmatrix}.$$

Le système intègre également un contrôleur flou qui va, au moyen de règles telles que

if temperature **is** Cold **then** Tension **is** High,

déterminer l'état requis pour le système de gestion du chauffage :

$$|\text{Chauff.}\rangle = \begin{bmatrix} 0.0 \; \textit{Low} \\ 0.1 \; \textit{Medium} \\ 0.9 \; \textit{High} \end{bmatrix}.$$

Cet état est enfin transformé en une valeur scalaire, imaginons la tension de la résistance et du ventilateur, au moyen du décodage que nous allons aborder par la suite.

Dans notre implémentation, nous avons choisi de travailler avec trois fonctions trapézoïdes. Un autre choix plus coûteux numériquement est l'emploi de sigmoïdes qui permettent d'introduire une nuance plus douce n'étant pas justifiée dans le cadre de cette implémentation. Nous

avons ainsi défini nos fonction d'encodage telle que

$$E_A : \mathbb{R} \to [0,1] \times [0,1] \times [0,1]$$
$$x \mapsto E_A(x) \triangleq |A\rangle = \begin{bmatrix} \frac{a_2 - x}{a_2 - a_1} \\ \frac{|x - a_2| - 1}{a_2 - a_3} \\ \frac{x - a_2}{a_3 - a_2} \end{bmatrix}.$$

Une telle fonction doit être définie par un vecteur **a** de trois éléments pour chaque variable que nous allons manipuler par la suite. Ces valeurs sont aussi désignées *valeurs caractéristiques des variables linguistiques* par la suite. L'allure de ce type de fonctions d'encodage est représentée pour trois variables à la Figure 3.1.

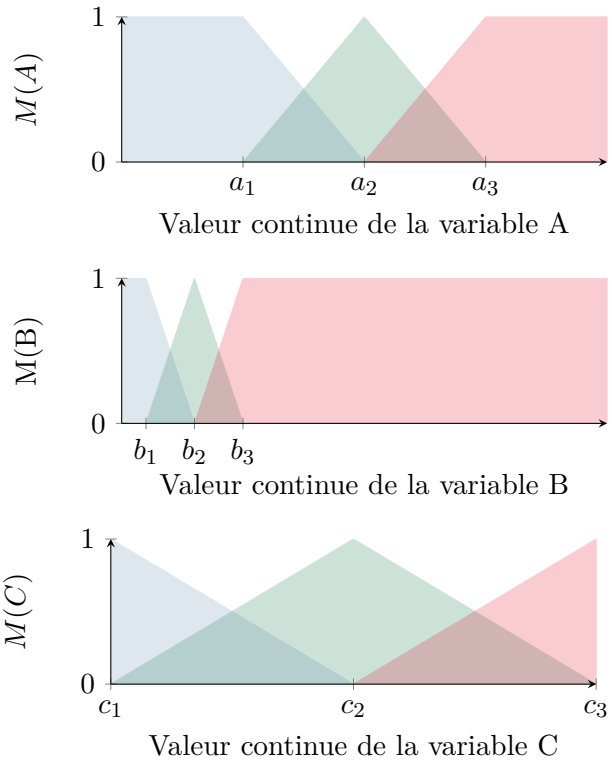


FIGURE 3.1 – **Exemple d'applications représentant chacune une variable.** Les trois valeurs linguistiques sont représentées en bleu, vert et rouge. Chaque fonction triangle donne l'appartenance à la valeur correspondante.

3.3 Règles de logique floue

Les règles régissant la logique floue découlent des règles de la logique classique en prenant soin de les généraliser au cas de valeurs de vérité réelles. Pour rappel

and	True	False
True	True	False
False	False	False

or	True	False
True	True	True
False	True	False

not	True	False
∅	False	True

Dans le cas de la logique floue, nous avons choisi de travailler avec les opérateurs de Zadeh présentés dans le Tableau 3.1. La mise en place de tels opérateurs consiste simplement en la substitution des opérateurs classiques par les opérateurs de Zadeh dans les propositions logiques. Ces opérateurs ont la propriété d'être équivalents aux opérateurs classiques pour des valeurs de vérité de $False = 0.0$ et $True = 1.0$.

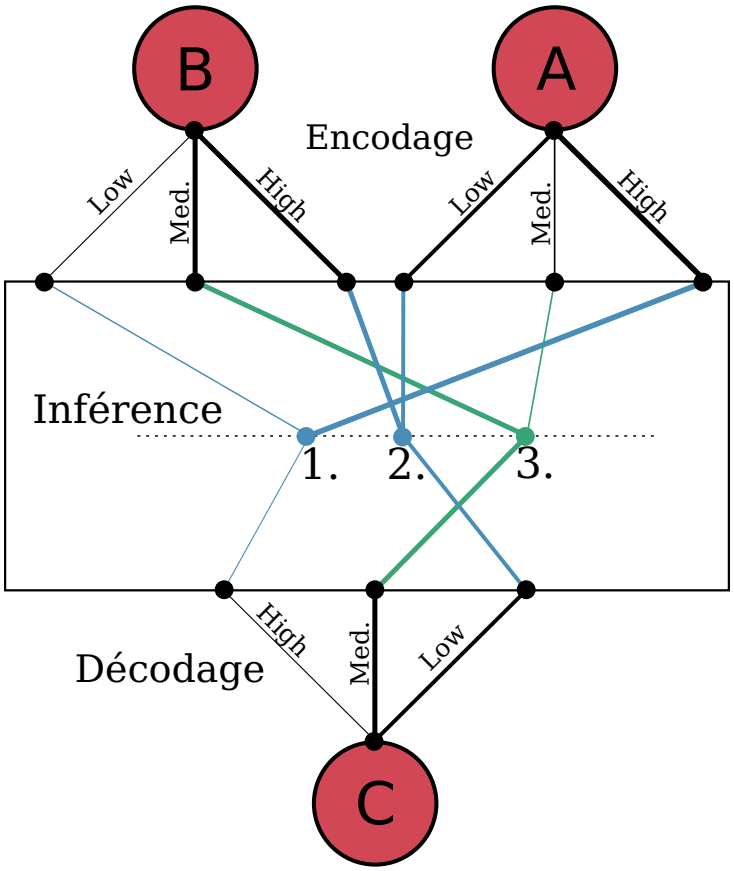


FIGURE 3.2 – **Vue schématique du processus d'inférence floue.** Processus complet d'inférence floue pour deux variables en entrée A et B ainsi qu'une variable contrôlée C. Les règles d'inférence sont symbolisées par les traits colorés traversant la ligne en pointillés. L'épaisseur des traits représente la valeur de l'appartenance. L'opérateur **and** est représenté en bleu et l'opérateur **or** en vert.

Booléen	Flou (Zadeh)
u **and** v	$\min[u, v]$
u **or** v	$\max[u, v]$
not v	$1 - v$

TABLE 3.1 – **Opérateur de Zadeh.** Correspondance entre les versions booléennes et floues des opérateurs fondamentaux que nous avons utilisés.

Les opérateurs de Zadeh vont ainsi permettre d'exprimer les règles de logique en opérations sur des réels.

Exemple

Interprétons la proposition suivante, règle **1.** à la Figure 3.2.

if A **is** High **and** B **is** Low **then** C **is** High. $(*)$

Nous commençons par lire le membre de droite par rapport à l'opérateur **then** qui va nous dire quelle valeur linguistique va être affectée, ici, il s'agit de HIGH pour la variable C. Nous noterons cette valeur déterminée par inférence $\bar{M}_{\text{High}}(C)$ étant donné qu'elle est associée à la fonction d'encodage $M_{\text{High}}(C)$. Maintenant, nous devons nous occuper du membre de droite de la proposition $(*)$, les propositions du type a **is** b doivent être subtituées par les valeurs prises par des fonctions d'appartenances $M_b(a)$. Soit

$$A \textbf{ is } \text{HIGH} \equiv M_{\text{HIGH}}(A)$$

et

$$B \textbf{ is } \text{LOW} \equiv M_{\text{LOW}}(B).$$

Cette valeur de vérité s'exprime enfin au moyen de l'opérateur de Zadeh **and** comme

$$v = \bar{M}_{\text{High}}(C) = \min\left[M_{\text{High}}(A), M_{\text{Low}}(B)\right].$$

3.4 Décoder une variable linguistique

Dans les sections précédentes, nous avons vu comment exprimer une variable continue sous la forme d'un état dans un univers de discours donné. Nous avons également expliqué l'adaptation des règles de logique classique à la logique floue. Nous abordons à présent l'opération de décodage qui va nous permettre de récupérer une valeur chiffrée à partir de l'état de sortie. Différentes méthodes permettent de procéder au décodage. Cette opération présente en entrée un état $|s\rangle$ représenté par un vecteur dans la base de variables linguistiques

$$|s\rangle = \sum_i^n \rho_i |l_i\rangle = \begin{pmatrix} \rho_1 \\ \vdots \\ \rho_i \\ \vdots \\ \rho_n \end{pmatrix},$$

la représentation sous forme de vecteur étant obtenue en identifiant notre base linguistique à une base canonique des réels de même dimension.

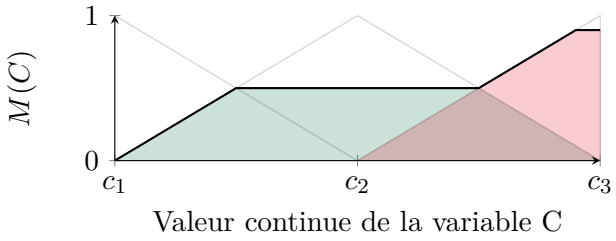


FIGURE 3.3 – **Application d'encodage floue modifiée pour tenir compte de l'état de sortie.** Les différentes fonctions d'appartenance ont été limitées par l'élément du vecteur d'état correspondant.

Il existe une vaste gamme de méthodes [18] pour mettre au point une application de décodage D_A telle que

$$\begin{aligned} D_A : [0,1]^n &\to \mathbb{R} \\ |s\rangle &\mapsto D_A(|s\rangle). \end{aligned}$$

Nous citerons par exemple la méthode du centroïde qui consiste à prendre pour valeur réelle le centroïde de la fonction résultante représentée en trait épais dans la Figure 3.3. Nous notons le centroïde $\langle x \rangle$ tel que

$$\langle x \rangle = \frac{\int x M_{res}(x) dx}{\int M_{res}(x) dx},$$

avec M_{res} la fonction maximum de toutes les fonctions d'appartenance qui s'écrit

$$\forall x, M_{res}(x) = \max\left(M_1(x), M_2(x), \ldots, M_n(x)\right).$$

Cette méthode est cependant coûteuse, car employée de manière intensive, chaque individu possédant plusieurs moteurs d'inférence floue. Nous avons ainsi choisi un modèle de décodage alternatif désigné inférence de Sugeno [19]. Ce modèle d'inférence plus simple formule la variable réelle décodée, notée x, de la manière suivante

$$x = \frac{\sum_{i=0}^{n} \rho_i a_i}{\sum_{i=0}^{n} \rho_i}.$$

Exemple

Un état est représenté à la Figure 3.3 pour notre variable C, il est défini comme

$$|C\text{hauff}\rangle = \sum_i^n \rho_i \, |l_i\rangle = \begin{bmatrix} 0.4 \; \textit{Low} \\ 0.4 \; \textit{Medium} \\ 0.2 \; \textit{High} \end{bmatrix}.$$

L'inférence de Sugeno nous donne comme valeur pour la tension du système de chauffage :

$$V = 0.4 \; c_1 + 0.4 \; c_2 + 0.2 \; c_3,$$

le vecteur $\mathbf{c}$ définissant la gamme de tension considérée LOW, MEDIUM ou HIGH.

Ceci conclut le chapitre présentant la logique floue. Nous avons découvert comment les règles logiques que nous allons énoncer dans le chapitre suivant sont implémentées en termes de contrôle sur base de fonctions d'encodage et de décodage. Nous avons également présenté les méthodes d'inférence et de décodage particulières employées dans notre implémentation ; méthodes résolument orientées vers la simplicité afin de respecter l'essence de la méthode à émergence faible qu'est la PSO.

Dans la partie suivante, nous allons découvrir la mise en œuvre des concepts présentés dans les trois chapitres précédents. Les variables observées et les paramètres contrôlés y seront définis et justifiés.

Deuxième partie

Méthode et implémentation

CHAPITRE 4

Contrôle de paramètres et d'hybridation

La motivation principale du travail a été présentée dans la partie précédente et trouve ses origines dans un énoncé du *No free lunch theorem* qui peut être résumé comme suit : une heuristique est adaptée à un certain ensemble de problèmes.

Notre objectif est donc de mettre au point un algorithme PSO s'adaptant au problème mais aussi à l'état de l'essaim. Pour y parvenir, nous avons exprimé de nombreux paramètres de la méthode comme fonction d'un contrôleur flou, lui-même paramétré par ce que nous désignerons comme les *métaparamètres* de l'algorithme. Dans le cadre de ce travail, les métaparamètres ont été ajustés manuellement mais aussi optimisés systématiquement. Nous avons cependant choisi de concentrer notre discours sur le choix manuel de ces paramètres.

Dans un premier temps, nous présentons la réflexion derrière un contrôle de paramètres sur un essaim de particules classiques. Ensuite nous présentons notre proposition d'hybridation dynamique entre deux variantes de la PSO, classique et quantique.

4.1 Principe du contrôle

Pour garantir la convergence de la PSO, un choix correct des paramètres est crucial de sorte qu'un certain intervalle légèrement dépendant du problème est permis pour ω, c_1 et c_2. Par exemple, une valeur de ω supérieure à 0.8 empêche généralement la convergence de l'essaim comme on peut le voir à la Figure 4.1. Cependant, une telle valeur pour ω peut être souhaitable au début de l'optimisation afin de maximiser l'exploration du domaine de recherche. Dès lors, si l'on souhaite exploiter des valeurs originales des paramètres, il est nécessaire de le faire avec les bonnes valeurs aux bons moments. Un tel contrôle de paramètres, statique, peut être obtenu à travers une fonction prenant pour argument le numéro de l'itération. Cependant un tel contrôle ad-hoc devra être modifié en fonction du cas et reste assez limité au niveau de son apport à l'émergence.

Par la suite, nous avons approché ce problème au moyen de contrôleurs en logique floue présentés à la section précédente, de manière similaire aux travaux de Marco S. Nobile [20]. Notre implémentation reste fidèle au principe d'émergence faible car chaque particule possède un contrôleur flou indépendant. Nous évitons ainsi un contrôle global nous faisant entrer dans l'émergence forte, conduisant facilement à une spécialisation de l'algorithme sur certaines fonctions. Nous désignerons cette spécialisation *overfitting*.

L'overfitting est un terme que nous empruntons ici à l'apprentissage machine afin de décrire une situation où la performance d'un algorithme est biaisée vers un cas test particulier. Ce comportement se produit en effet si l'on introduit un comportement passant outre l'émergence dans l'essaim. Nous avons ainsi effectivement au sein de chaque particule une fonction non-linéaire de différentes variables caractérisant son état instantané.

Dans la suite, nous allons présenter les variables sondes, inspirées [1] de [20] et les variables que nous avons choisi de contrôler.

4.2 Définition de sondes floues

Nous présentons dans cette section les variables que nous employons pour diriger le contrôleur de paramètres. Les fonctions d'appartenance sont les trois fonctions triangles présentées dans le chapitre précédent. Les vecteurs de paramètres associés **a** sont listés à la fin de ce chapitre.

4.2.1 Proximité à l'attracteur topologique

La première variable est la distance de la particule à son meilleur ami, c'est-à-dire à la meilleure particule de son voisinage. Celle-ci est donnée par

$$\delta = \frac{||\mathbf{x} - \mathbf{t}||}{||\mathbf{d}||},$$

où $\mathbf{x}$ représente la position de la particule, $\mathbf{t}$ est la meilleure position connue dans son voisinage et $\mathbf{d}$ est le vecteur diagonal du domaine de recherche ; cette quantité est ainsi comprise dans l'intervalle $[0, 1]$. Cette première quantité nous donne une information sur la distance de la particule par rapport à son voisinage actuel. La variable prend trois valeurs linguistiques : NEAR, MID et FAR.

4.2.2 Activité de l'agent

La seconde grandeur observée, notée ϕ est une mesure de l'activité de l'agent. Elle se compose de deux facteurs, d'une part la variation d'aptitude et d'autre part, la norme de la vitesse. La norme de la vitesse est distribuée par rapport à la dimension de l'espace de recherche, ainsi

$$\phi|_{t+1} = \frac{||f(\mathbf{x})|_t - f(\mathbf{x})|_{t-1}||}{f_{worst}|_T - f_{best}|_T} \frac{||\mathbf{v}|_t||}{||\mathbf{d}||}.$$

Cette quantité est comprise entre -1 et 1. La notation T signifie que l'évaluation de ces grandeurs se fait périodiquement, en pratique, toutes les 100 itérations. La variable prend trois valeurs linguistiques : WORSE, SAME et BETTER.

4.2.3 Temps de simulation

Cette variable prend une place assez particulière au sein de nos contrôleurs car il s'agit d'une propriété globale et donc d'un contrôle global. Nous explorons avec cette variable l'effet d'un contrôle externe pour d'une part amener de l'émergence forte au niveau du taux de connexion du graphe topologique et d'autre part pour servir de garde-fou à certains comportements nuisibles en début et fin de simulation.

1. Certaines adaptations ont été effectuées pour adapter les définitions à une topologie moins connectée. Les définitions des références d'aptitude sont également modifiées

Cette variable est simplement formulée comme

$$\tau|_t = \frac{t}{t_{max}}$$

et prend trois valeurs linguistiques : STARTING, HALFWAY et ENDING.

4.3 Les variables à contrôler

Le choix des paramètres à contrôler est une étape importante dans la mise au point d'un essaim *proactif*. Appliquer une variation sur trop de paramètres pourrait conduire à des redondances inutiles ou pire, à une diminution de l'efficacité de l'essaim si certains comportements imprévus et néfastes émergeaient dans la complexité du contrôle. Nous souhaitons également conserver le principe d'émergence faible. Pour cela, nous devons agir au niveau des individus et créer une réflexion locale plutôt qu'un contrôle global. Par comparaison, notre objectif est de donner plus de perception aux particules au lieu de les trainer artificiellement jusqu'à la solution. Les paramètres que nous allons présenter par la suite sont choisis pour intégrer les apports de certaines variantes telles que la *Linearly Decreasing Inertia PSO* [21] dans un cadre souple et respectueux de l'émergence faible.

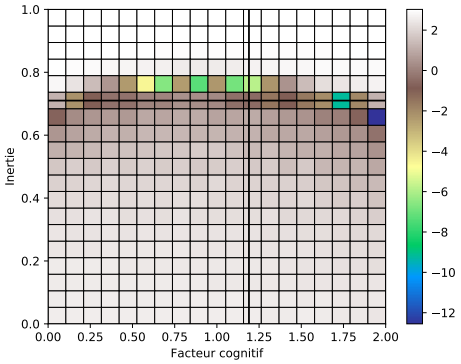
4.3.1 Contrôle de l'inertie

L'inertie est un paramètre capital définissant le comportement de l'essaim. Il est responsable de l'exploration en permettant à une particule de rester plus longtemps sur sa trajectoire sous pression sociale ou cognitive. Une étude préliminaire a confirmé ces affirmations[17]. Nous pouvons observer à la Figure 4.1a que l'inertie conditionne la performance de la PSO, une observation similaire à celle qui a motivé sa mise au point par Shi, Yuhui et Eberhart [14].

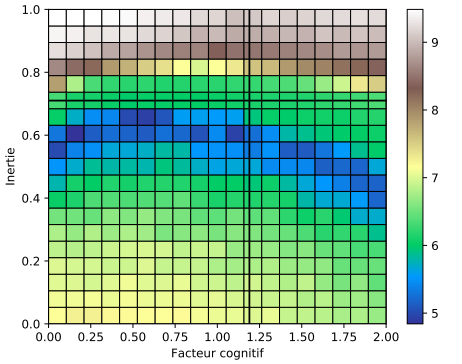
Les règles de logique présentées dans l'Algorithme 1 visent à implémenter deux comportements. D'une part, une particule diminuant son aptitude et isolée va perdre en inertie, rejoignant ainsi plus facilement son voisin ou son meilleur souvenir. D'autre part, une particule qui vient d'améliorer son aptitude ou qui est trop proche de son meilleur ami va tendre à continuer son exploration.

Data : Encoded ϕ and δ
Result : Swarm with adapted parameters
for particles *in* swarm **do**
  if ϕ *is* WORSE *or* δ *is* FAR **then**
    set ω to LOW ;
  end
  if ϕ *is* SAME *or* δ *is* MID **then**
    set ω to MEDIUM ;
  end
  if ϕ *is* BETTER *or* δ *is* NEAR **then**
    set ω to HIGH ;
  end
end

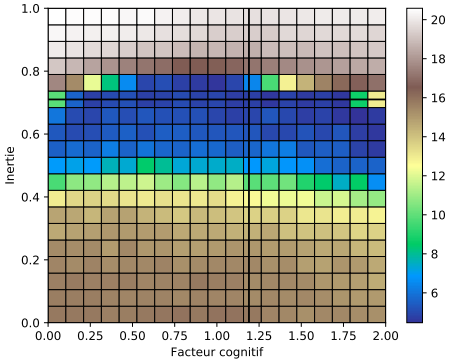
Algorithme 1 : Règles de logique floue pour la valeur de l'inertie.



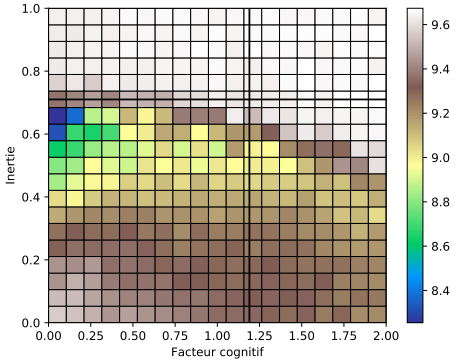
(a) Ackley.



(b) Rastrigin.



(c) Rosenbrock.



(d) Schwefel.

FIGURE 4.1 – **Aptitude finale (Log10) sur différents cas test pour une variation des paramètres inertie et cognitif à coefficient social constant.** Les valeurs conseillées dans la littérature pour ces deux paramètres sont représentées en traits noirs. Par aptitude finale, nous entendons la moyenne de la meilleure aptitude obtenue sur 15 exécutions indépendantes de l'algorithme d'optimisation.

4.3.2 Contrôle de la vitesse

Un contrôle sur la vitesse est appliqué au moyen de bornes inférieures et supérieures, à chaque itération de l'optimiseur. La condition suivante est vérifiée pour chaque particule avant son déplacement :

$$\forall i = 1 \ \dots \ n, \ m_i < x_i < M_i,$$

avec

$$\mathbf{m} = \sigma_l \mathbf{d}, \ \mathbf{M} = \sigma_L \mathbf{d}.$$

Nous avons donc $\mathbf{m}$ et $\mathbf{M}$ les vecteurs minorant et majorant la vitesse. Les paramètres soumis à un contrôle flou sont σ_l et σ_L suivant les règles d'inférence définies dans les algorithmes 2 et 3.

Ces deux contrôles ont la vocation d'améliorer les mouvements de l'essaim en assurant sa mobilité dans des zones où l'aptitude est mauvaise et en le ralentissant dans les régions intéressantes.

Data : Encoded ϕ and δ
Result : Swarm with adapted parameters
for particles *in* swarm **do**
  if ϕ *is* SAME *or* ϕ *is* BETTER *and* δ *is* FAR **then**
    set σ_l to LOW ;
  end
  if ϕ *is* SAME *or* δ *is* NEAR **then**
    set σ_l to MEDIUM ;
  end
  if ϕ *is* WORSE **then**
    set σ_l to HIGH ;
  end
end

Algorithme 2 : Règles de logique floue pour la borne inférieure de la vitesse σ_l.

Data : Encoded ϕ and δ
Result : Swarm with adapted parameters
for particles *in* swarm **do**
  if ϕ *is* SAME **then**
    set σ_L to LOW ;
  end
  if ϕ *is* SAME *or* ϕ *is* BETTER *or* δ *is* NEAR **then**
    set σ_L to MEDIUM ;
  end
  if ϕ *is* WORSE *or* δ *is* FAR **then**
    set σ_L to HIGH ;
  end
end

Algorithme 3 : Règles de logique floue pour la borne supérieure de la vitesse σ_L.

4.3.3 Contrôle de l'hybridation

Notre PSO hybride consiste à assigner deux comportements différents à chaque particule. A chaque itération, une particule a le choix entre un comportement quantique et un comportement classique. Cette hybridation vise à tirer parti des avantages à la fois de la PSO quantique et de la PSO classique. La PSO quantique a une performance intéressante sur bon nombre de cas. De plus, grâce à son caractère discontinu et stochastique, elle apporte une méthode de

recherche assez différente de la PSO classique. Cependant, une implémentation efficace ne peut que difficilement passer par un paramètre statique qui déterminerait la probabilité de chaque comportement. Nous verrons par la suite qu'un contrôle statique conduit à une moins grande polyvalence de l'algorithme.

Nos règles de contrôles sont résumées dans l'Algorithme 4. Nous cherchons à bénéficier des capacités d'exploration de la PSO quantique quand l'algorithme démarre et aussi quand les particules sont éloignées de leur meilleur ami, ce qui est le cas quand la topologie est réévaluée ou quand aucune solution dominante n'est dégagée. Nous permettons ainsi aux particules d'explorer une vaste zone autour du centre des points **t** et **p**.

Au contraire, si la particule est en train de trouver un minimum, la PSO quantique (QPSO) pourrait la priver de l'exploitation si son meilleur ami ou son meilleur souvenir est lointain. Nous diminuons donc le taux de PSO quantique.

Enfin, si la particule est proche de son meilleur ami, le potentiel quantique cause une distribution très étroite, ce qui risque de piéger les particules, nous diminuons donc l'hybridation afin de laisser le contrôleur inertiel 1 et de vitesse 2 faire leur travail avec la PSO canonique.

Data : Encoded ϕ and δ
Result : Swarm with adapted parameters
for particles *in* swarm **do**
  if ϕ *is* BETTER *or* δ *is* NEAR **then**
    set h to LOW ;
  end
  if δ *is* MID *or* t *is* HALFWAY **then**
    set h to MEDIUM ;
  end
  if δ *is* FAR *or* t *is* STARTING **then**
    set h to HIGH ;
  end
end

Algorithme 4 : Règles de logique floue pour la probabilité d'adopter un comportement quantique h.

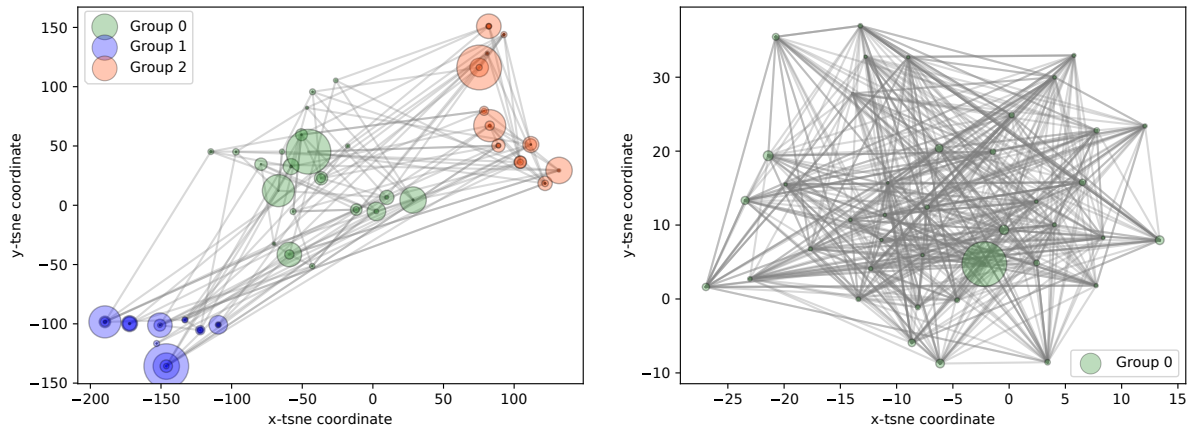
4.3.4 Contrôle de la topologie

Le taux de connexions de la topologie ainsi que sa forme jouent un rôle capital dans le comportement de l'essaim. Si nous prenons pour exemple la topologie totalement connectée, la circulation globale et instantanée de l'information entre les particules conduit à une convergence rapide de l'algorithme dans un minimum local. Au contraire, une topologie faiblement connectée, par exemple avec $\mathcal{K} = 3$ pour $n_S = 40$ conduit à une grande autonomie des particules qui échangent de l'information plus lentement et dans des réseaux parfois disjoints les uns des autres, comme illustré dans la Figure 4.2a. Dans cette Figure, on illustre un nombre de connexions plus faible avec $\mathcal{K} = 3$ que dans 4.2b où $\mathcal{K} = 15$. La topologie plus connectée a forcé l'essaim à s'agglomérer autour d'une solution optimale tandis que la topologie plus faible a conduit à la création de divers groupes autour de solutions prometteuses.

Les visualisations sont obtenues par la méthode t-SNE qui ne conserve ni les distances ni les positions, mais uniquement la distribution. La forme de ces graphes est influencée à la fois par la fonction considérée et la topologie. Il s'agit d'un moyen pratique afin d'évaluer l'interaction entre la topologie et l'essaim dans un espace à haute dimension. Le paramètre de perplexité pour la t-SNE est de 3 et le rayon ϵ de DBSCAN[2] a été déterminé par *grid search* afin que

2. DBSCAN est un algorithme de classification par clustering tel que k-means. l'algorithme k-means travaille itérativement en décrivant le clustering par la norme euclidienne à partir de centroïdes. DBSCAN travaille en

tous les points soient classifiés, il s'agit donc du rayon minimal incluant tous les points dans la classification.



(a) Graphe faiblement connecté, Rosenbrock (b) Graphe fortement connecté, Ackley

FIGURE 4.2 – **Visualisation d'un graphe topologique via t-SNE et DBSCAN.** Le graphe topologique est obtenu par projection t-SNE de l'essaim hybride sur les cas test Rosenbrock et Ackley à 50 dimensions. Les différents groupes sont distingués en couleurs par l'algorithme DBSCAN.

Comme indiqué dans la partie précédente, nous avons choisi de travailler avec une topologie faiblement connectée adaptative. Cette variante permet de rétablir une communication globale de l'information au sein de l'essaim sans augmenter le taux de connexion. Ce résultat est obtenu en réévaluant le graphe topologique si la meilleure solution de l'essaim stagne.

Enfin, notre implémentation du contrôle topologique est liée au nombre d'itérations effectuées par rapport au buget total, ceci afin de basculer d'une PSO exploratoire vers une recherche locale sur un même processus d'optimisation. L'Algorithme 6 décrit ce contrôle assez simple. La topologie finale est semblable à celle présentée dans la Figure 4.2b, l'unification du réseau de communication pousse l'essaim à se regrouper sur les dernières itérations. L'homogénéité du graphe b. signifie, en gardant à l'esprit que seule la séparation des individus est à interpréter en t-SNE, que l'essaim est distribué de manière uniforme dans l'espace de recherche. Le rayon moyen de l'essaim dans l'espace de recherche, que l'on écrit

$$r_{swarm} = \text{mean}_{k=1}^{n_S}(||\mathbf{c} - \mathbf{x}_k||),$$

où $\mathbf{c}$ est le centre de l'essaim (position moyenne) nous indique s'il s'agit ou non d'une agglomération.

propageant les classes de proche-en proche et est ainsi adapté aux distributions uniformes possédant des formes longilignes ou concaves. Le paramètre ϵ est le rayon de propagation d'une particule.

Data : Encoded ϕ and δ
Result : Swarm with adapted parameters
for particles *in* swarm **do**
  if δ *is* NEAR *or* t *is* STARTING **then**
    set $\mathcal{K}$ to LOW ;
  end
  if δ *is* MID *or* t *is* HALFWAY **then**
    set $\mathcal{K}$ to MEDIUM ;
  end
  if δ *is* FAR *and* t *is* ENDING **then**
    set $\mathcal{K}$ to HIGH ;
  end
end

Algorithme 5 : Règles de logique floue pour le nombre de voisins $\mathcal{K}$.

Cependant, la topologie aléatoire adaptative reste orientée sur une exploration de l'espace de recherche et ralentit l'exploitation. Nous avons donc souhaité mettre au point un contrôle du nombre de voisins en fonction de l'itération. Ceci afin d'augmenter la localisation de la recherche en augmentant le taux de connexion à la fin du processus d'optimisation.

4.3.5 Contrôle de la localisation

La PSO quantique (QPSO) possède un unique paramètre g qui définit la portée des sauts quantiques. La réflexion derrière ce contrôleur est très proche de cette qui dirige l'hybridation. En effet, la portée de l'exploration est inversement proportionnelle à g.

Data : Encoded ϕ and δ
Result : Swarm with adapted parameters
for particles *in* swarm **do**
  if δ *is* NEAR *or* t *is* STARTING **then**
    set g to LOW ;
  end
  if δ *is* MID *or* t *is* HALFWAY **then**
    set g to MEDIUM ;
  end
  if δ *is* FAR *and* t *is* ENDING **then**
    set g to HIGH ;
  end
end

Algorithme 6 : Règles de logique floue pour la localisation g.

4.4 Paramétrisation des contrôleurs

4.4.1 Paramétrisation manuelle

Pour conclure ce chapitre, nous présentons les métaparamètres **a** employés dans notre implémentation. Les valeurs présentées dans le tableau 4.1 ont été inspirées par les références [20] et [6] et modifiées en accord avec notre topologie faiblement connectée.

Les valeurs sont généralement prises autour de celles conseillées dans la littérature, dans le cas du nombre d'amis, nous allons jusqu'à autoriser 15 amis, ce qui donne une topologie extrêmement connectée.

Nom	Symbole	Bas	Moyen	Haut
Inertie	ω	0.3	0.5	1.0
Vitesse minimale	σ_l	0.0	0.001	0.01
Vitesse maximale	σ_L	0.4	0.5	0.6
Hybridation	h	0.0	0.5	1.0
Localisation	g	0.1	1.0	2.0
Amis	$\mathcal{K}$	3	8	15

TABLE 4.1 – **Métaparamètres définissant la forme des fonctions d'appartenance.**

4.4.2 Optimisation des paramètres

La détermination des paramètres des fonctions d'appartenance floues est une tâche complexe car ils sont nombreux et interdépendants. En effet, la modification d'une partie des lois logiques implique un renouvellement des paramètres à cause de l'interdépendance. Dans le cadre de ce travail, nous tenons à mentionner la piste de l'optimisation systématique des contrôleurs flous qui a été brièvement explorée.

Nous posons un problème d'optimisation sur l'espace des métaparamètres S_m, pour lesquels nous définissons des extrema raisonnables sur base de notre paramétrisation manuelle.

Une manière de formuler une fonction d'aptitude représentant la performance des métaparamètres est de choisir un vaste ensemble de fonctions test $f_i(\mathbf{x}), i = 1, \dots N$ et d'évaluer le résultat moyen de l'algorithme d'optimisation sur $r = 5$ exécutions pour différentes valeurs des métaparamètres $\mathbf{a}$, noté $\bar{f}_i^{best}(\mathbf{a})$. La fonction objectif de notre problème de méta-optimisation s'exprime alors comme

$$f_{comp}(\mathbf{p}) = \sum_{i=i}^{N} \sigma(\bar{f}_i^{best}(\mathbf{a})),$$

où σ est une fonction visant à équilibrer les valeurs d'aptitude de manière continue, par exemple une sigmoïde définie comme

$$\sigma(x) = \frac{x}{1 + |x|}.$$

Cet équilibrage est requis afin d'égaliser les contributions des différentes fonctions à l'aptitude résultante.

Le problème ainsi posé peut être optimisé pour obtenir des valeurs de métaparamètres performants sur une gamme robuste de problèmes. Malheureusement, l'évaluation de f_{comp} est assez coûteuse et une optimisation rigoureuse employant un métamodèle n'a pas pu être réalisée dans les contraintes de temps de ce mémoire. Des résultats préliminaires basés sur trois fonctions au profil distinct (Schwefel, Ackley et Rosenbrock) nous indiquent cependant que cette méthode de métaparamétrisation est prometteuse.

4.4.3 Une piste alternative : réseau neuronal

L'optimisation systématique des contrôleurs flous comme présentée dans la section précédente concerne uniquement les métaparamètres. Afin d'optimiser totalement un système de contrôle complexe, il devient nécessaire de modifier les règles de logique. Or, nos règles de logique ne sont pas pratiques à implémenter dans un processus d'optimisation. Il s'agit d'une interface humaine qui était très utile dans ce travail.

Pour supporter nos développements futurs, nous nous tournerons vers l'emploi d'un réseau neuronal pour établir le lien entre les capteurs et les paramètres à contrôler, comme illustré à la Figure 4.3. Le réseau serait ainsi entrainé sur un vaste ensemble de fonctions et problèmes de référence par un autre algorithme d'optimisation afin d'obtenir un comportement robuste et efficace.

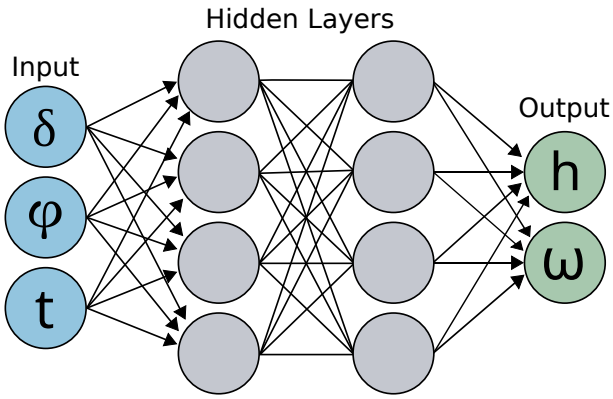


FIGURE 4.3 – **Principe d'un réseau de neurones comme contrôleur.** Les sondes sont présentées en entrée du réseau, en bleu. Les couches masquées du réseau présenteraient une profondeur assez faible afin d'atteindre une complexité similaire à celles des contrôleurs flous. Les divers paramètres se retrouveraient en sortie de ce réseau, ici en vert.

Cette représentation alternative du contrôleur de paramètres serait plus facile à optimiser étant donné le formalisme et les librairies déjà en place. Nous y perdons certes l'interface humaine que représentent les règles de logique mais si le nombre de paramètres et de sondes croît, il devient nécessaire d'employer une méthode d'optimisation systématique. Un tel réseau de neurones devrait rester peu profond selon l'expérience acquise dans ce travail, afin de ne pas être suradapté sur l'ensemble de fonctions. Les contrôleurs doivent donner de bonnes tendances aux paramètres mais ne doivent pas rendre la particule trop intelligente au risque de détruire la cohésion de l'essaim et par là même l'émergence.

Nous arrivons désormais au terme de ce chapitre consacré à la description des contrôleurs flous employés. Ces contrôleurs se composent de variables mesurées et encodées et de paramètres contrôlés au moyen d'énoncés logiques. Nous avons enfin fourni les valeurs de métaparamètres définissant le profil des fonctions d'encodage qui ont été déterminées empiriquement avec une volonté de progresser vers une optimisation systématique.

CHAPITRE 5

Implémentation et difficultés rencontrées

Dans les chapitres précédents, nous avons présenté les différents concepts que nous souhaitions mettre en oeuvre dans notre algorithme. Dans ce chapitre, nous nous attarderons sur les difficultés techniques rencontrées lors de la programmation desdits concepts. Les deux grandes thématiques seront ainsi la logique floue et le design *Entity Component System* (ECS).

5.1 ECS, un design adapté aux populations artificielles ?

Les principes du design ECS en programmation trouvent leur origine dans le développement de jeux vidéos. On peut s'étonner de l'application d'un tel design pour un algorithme évolutionnaire jusqu'à ce que l'on se penche sur les similitudes existant entre les deux systèmes. Nous allons cependant commencer par décrire le concept de design ECS.

5.1.1 Justification du paradigme

L'ECS est une architecture de développement mise au point en réaction à certains défauts de la programmation orientée objet (POO), notamment

- la complexité grandissante des graphes d'héritage ;
- l'obscurcissement du code à travers plusieurs couches d'encapsulation ;
- l'incohérence du cache quand il s'agit d'itérer sur les objets ;
- les cascades de destructeurs virtuels.

Dans des applications devant travailler en temps réel tout en supportant des simulations hétéroclites et gourmandes (intelligence artificielle, physique, ...), ces défauts deviennent insurmontables et freinent le développement et parfois même les performances de l'application.

Le design ECS quant à lui apporte une réponse à ces problèmes en suivant deux principes [22], désignés

- *composition over inheritance* ;
- *data-oriented design.*

Le premier principe consiste à favoriser la conception d'un nouvel objet à travers le regroupement d'objets élémentaires au lieu de le créer à partir d'un parent. Ce principe vise à limiter la quantité de classes et leurs liens de dépendance en poussant le programmeur à former des

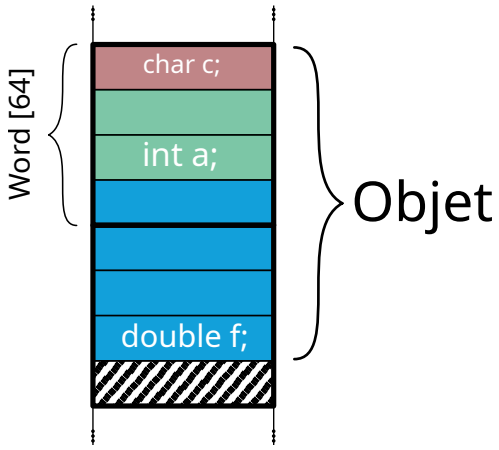


FIGURE 5.1 – **Illustration d'une mémoire non alignée.** Un segment de la mémoire RAM est représentée. Les contours des *mots* sont représentés en gras, les différentes variables sont représentées en couleurs.

classes telles que des briques élémentaires à la responsabilité unique. Le choix de ce principe permet donc d'isoler facilement les différentes responsabilités au sein du programme tout en gardant une implémentation transparente et simple.

Le second principe, *data-oriented design* trouve encore une fois son origine dans la création de jeux vidéos, en particulier sur consoles où les ressources sont fortement limitées. Ce principe de design consiste à porter une grande attention à l'emploi du cache du processeur, c'est-à-dire un espace mémoire plus rapide que la mémoire RAM disposée juste avant les registres typiquement manipulés en assembleur.

Cette approche ECS se concentre donc sur la mise au point de structures de données optimisées en fonction de la fréquence de leur emploi et de l'ordre dans lequel elles sont accédées, ceci afin de limiter le nombre de *cache misses*. Un *cache miss* est une situation où un segment de données requis ne se trouve pas dans la mémoire cache et doit être du depuis une mémoire plus lente telle que la RAM ou un autre niveau de cache. Un cas illustratif d'une structure provoquant ces ratés est représenté à la Figure 5.1.

Si nous souhaitons lire la variable **double** f, le coût associé sera doublé car le processeur doit lire les deux blocs de mémoire entourés en gras, décaler le premier de trois fois 16bits vers le haut et le second de 16bits vers le bas, ensuite une opération **or** sera appliquée pour enfin obtenir le résultat.

Ce type de comportement se retrouve à différents niveaux dans le fonctionnement du processeur et de la mémoire en particulier. Une manière d'en limiter les effets négatifs est d'avoir une mémoire homogène. Cette précaution n'est pas compatible avec les principes de la POO qui pousse à employer des structures de données hétérogènes. Par exemple, un objet *particule* dans une simulation n-corps possède un nom, une position et un poids, soit une structure hétérogène similaire à la Figure 5.1 si aucune précaution n'est prise. La structure s'écrit :

```
struct Particule
{
  double x, y, z;
  double w;
  string name;
};

// Une particule est un objet intuitif dans l'essaim.
Particule particules[100];
```

Si nous sommes amenés à effectuer une opération récursive sur les positions, comme typi-

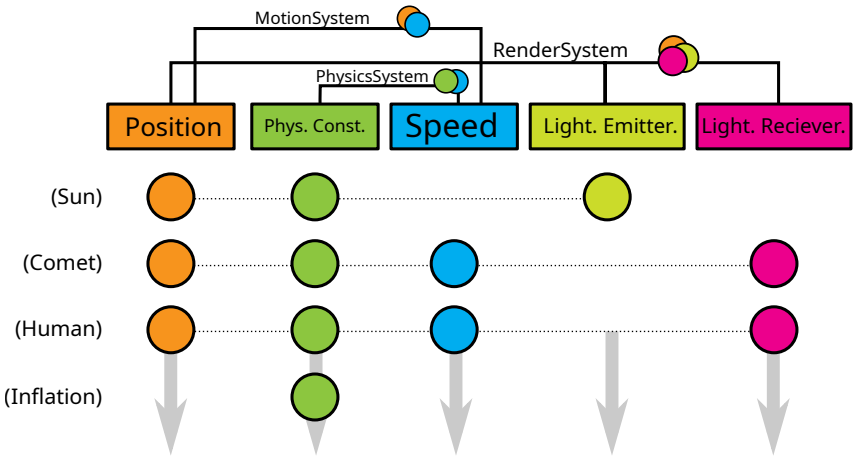


FIGURE 5.2 – **Diagramme ECS pour notre illustration nbody.** Les différents vecteurs de composants sont représentés par les colonnes, les lignes représentent des entités particulières. Les systèmes sont indiqués ainsi que les composants sur lesquels ils itèrent. Les flèches grises symbolisent les différents flux de données.

quement un calcul de norme euclidienne, nous serons bien plus efficaces en travaillant avec une structure du type :

```
string nom_particule[100];
double x_particule[100];
double y_particule[100];
double z_particule[100];
double w_particule[100];

// Une particule est donc un simple indice a manipuler.
int particule = 3;
```

Dans cette formulation cependant, il n'est plus possible de bénéficier de la formulation en forme de structure. Le code a perdu en abstraction, ce qui peut être un problème si l'on préfère cette manière de travailler.

5.1.2 Entités, composants et systèmes

Le paradigme ECS découpe la conception d'un programme ou d'une partie d'un programme en trois domaines : les entités, les composants et les systèmes. Les composants sont nos structures de données, ils sont stockés entre composants du même type dans des vecteurs contigus ou optimisés en fonction de leur emploi. Les composants ne contiennent aucune logique. Au contraire, les systèmes contiennent toute la logique du programme, regroupée par thématique. Chaque système a accès à certains composants uniquement, comme représenté à la Figure 5.2. Cette architecture, en plus de clarifier la séparation des responsabilités, permet de faciliter la mise en parallèle des systèmes ainsi formés. On peut en effet, dans notre exemple, isoler le système de rendu sur un fil d'exécution différent du reste du programme de simulation et ceci sans effectuer de changements majeurs au niveau du code.

5.2 Implémentation de la logique floue

Dans cette section, nous présentons le second défi rencontré lors de l'implémentation de notre algorithme qui concerne la description du système de contrôle en logique floue. Nous souhaitons mettre au point un niveau d'abstraction suffisant afin de formuler les règles d'inférence floue de manière lisible tout en gardant des performances optimales étant donné que ces contrôleurs sont évalués de manière intensive.

Pour réaliser cette implémentation, nous avons exploité les capacités de surcharge d'opérateurs du C++ ainsi qu'une structure orientée objet permettant de représenter les propositions logiques par des arbres binaires.

5.2.1 Écriture d'un contrôleur flou

Nous définissons lors de l'initialisation de l'algorithme les variables floues et leurs fonctions d'appartenance triangulaires comme définies dans le chapitre précédent :

```
// Fuzzy variables or 'probes'
FuzzyVar A1(0.1 /** small */, 0.5 /** medium */, 0.9 /** high */);
FuzzyVar A2(0.001, 0.01, 0.1);
FuzzyVar A3 ...;
// Fuzzy parameters or controlled variables
FuzzyVar C ...;
```

Ensuite, pour une question de performance, les règles de logique sont aussi définies lors de l'initialisation, afin d'être stockées dans des arbres syntaxiques binaires que nous allons présenter par la suite.

Nous traduirons ici en C++ les deux règles s'énonçant

if A1 **is** HIGH **or** A2 **is** LOW **then** C **is** LOW

et

if A1 **is** LOW **or** (A2 **is** HIGH AND A3 **is** HIGH) **then** C **is** HIGH.

Le formalisme suivant est permis par la surcharge d'opérateurs renforcée de polymorphisme du C++ :

```
Rule rule1 = (C.low << (A1.high || A2.low));
Rule rule2 = (C.high << (A1.low || (A3.high & A2.high)));
RuleSet controller = (rule1, rule2);
```

Les règles de logiques sont stockées dans un arbre syntaxique binaire tel que celui représenté à la Figure 5.3. Chaque élément de ce graphe est un objet de type nœud. Un nœud possède la capacité polymorphe d'être évalué en tant qu'un nombre réel. Les feuilles de cet arbre sont une forme particulière de nœuds qui s'évaluent au moyen de fonctions d'appartenance. Les nœuds communs quant à eux possèdent toujours deux noeuds subalternes qui sont évalués avant le nœud parent ; le nœud parent étant responsable de l'application des opérateurs de Zadeh. Dans le cas de l'opérateur **not**, la référence vers le second nœud est nulle. Le nœud parent possède l'information sur le type d'opérateur.

Ainsi, l'évaluation se propage en cascade depuis les sommets de l'arbre et les valeurs d'appartenance remontent depuis les feuilles vers le sommet.

Maintenant que les 2 règles et 4 variables sont exprimées, la suite est assez systématique, à chaque évaluation du contrôleur, des états sont générés à partir de variables-sondes et de leur valeur continue respective, par exemple

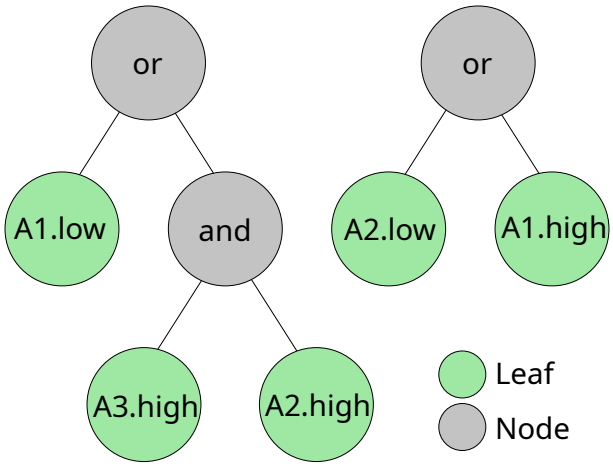


FIGURE 5.3 – **Représentation d'une règle de logique sous forme d'arbre syntaxique binaire.** Tous les nœuds du graphe héritent d'une classe *nœud* commune qui leur donne la propriété de pouvoir être évalué à un moment donné pour livrer un réel, valeur d'appartenance.

```
State state1 = (A1 == 0.42);
State state2 = (A2 == ...);
State state3 = (A3 == ...);
```

et enfin, la valeur réelle du paramètre à contrôler est obtenue par application des règles logiques suivie du décodage de l'état de sortie :

```
// Applying fuzzy rules
State stateC = controller.apply({ state1, state2, state3 });
// Decoding stateC
double valueC = C.decode(stateC);
```

Ceci conclut ce chapitre consacré aux difficultés d'implémentation. Nous avons présenté l'emploi de deux paradigmes de programmation qui nous ont chacun facilité le développement d'une partie de l'iplémentation : l'ECS pour la gestion de populations et la POO pour l'abstraction de règles de logique floue.

Dans le chapitre suivant, nous allons décrire les performances des algorithmes ainsi implémentés.

Troisième partie

Performance de l'algorithme

CHAPITRE 6

Comparaison des différentes variantes

Dans ce chapitre, nous allons présenter les performances obtenues par nos différentes variantes de l'algorithme PSO. L'évaluation des performances d'un algorithme heuristique est une tâche particulièrement délicate car il s'agit du seul moyen de caractériser sa convergence, contrairement aux méthodes exactes, qui sous certaines hypothèses, offrent analytiquement une garantie de convergence. Il est ainsi nécessaire de tester chaque algorithme sur un ensemble de fonctions présentant des difficultés diverses à multiples reprises afin de tenir compte du caractère stochastique de ces méthodes.

6.1 Expérience numérique

Afin de mettre au point une comparaison significative, nous avons employé un ensemble de fonctions assez vaste, rassemblant des profils unimodaux, multimodaux,... Une description de ces fonctions est disponible dans l'annexe A, accompagnée de leur profil et de leur domaine de définition. Le Tableau 6.1 reprend leurs caractéristiques principales.

Identifiant	Dimensions	Multimodale	Intervalle
Ackley	50		$[-32.775, 32.775]$
Rastrigin	50		$[-5.12, 5.12]$
Rosenbrock	50		$[-2.048, 2.048]$
Alpine1	50		$[0.0, 10.0]$
Alpine2	50		$[0.0, 10.0]$
Schwefel	50		$[-500.0, 500.0]$
Griewank	50		$[-512.0, 512.0]$
Qing	50		$[-500.0, 500.0]$
Michalewicz	50		$[0.0, \pi]$
Xinsheyang1	50		$[-5.0, 5.0]$
Xinsheyang2	50		$[-6.2832, 6.2832]$
Stiblinskitank	50		$[-5.0, 5.0]$

TABLE 6.1 – **Résumé des fonctions.** La modalité, le nombre de variables et le domaine sont indiqués.

La sélection de fonctions visant à évaluer l'algorithme n'a pas été faite à priori mais correspond à notre volonté de produire un algorithme robuste sur de multiples profils de fonctions.

Nous sommes ainsi partis d'un petit ensemble de fonctions que nous avons étoffé au fur et à mesure du développement afin d'identifier les forces et les faiblesses de notre modèle.

Les données présentées par la suite résultent du processus de minimisation des fonctions présentées en annexe. Toutes ces fonctions ont été adaptées afin que leur minimum soit nul. Les résultats sont obtenus en sessions de 1000 itérations avec une population de 40 particules pour toutes les variantes. Le nombre d'évaluations de fonctions s'élève donc à 40000 pour une session complète. Afin de réduire l'influence des différents comportements stochastiques présents dans les méthodes employées, nous effectuons $n_{\text{stat}} = 100$ exécutions de l'algorithme avec n_{stat} graines aléatoires différentes pour tous les graphes présentés. Ceci afin de pouvoir assimiler le comportement moyen à celui de l'algorithme. Nous avons pour cela représenté les courbes de convergence avec un intervalle de confiance de 99.7% qui s'écrit

$$I_c = 3\sigma/\sqrt{n_{stat}}.$$

où σ est l'écart type. Les cas tests sont évalués dans un espace à 50 dimensions sur le sous-espace de $\mathbb{R}^n$ associé à la fonction, comme spécifié en annexe.

6.2 Contrôle de paramètres en logique floue

Dans un premier temps, nous avons souhaité évaluer différents types de contrôle de paramètres, pour la plupart déjà présents dans la littérature. Nous avons donc combiné les contrôles sur l'inertie, sur les bornes de la vitesse et sur le facteur social comme décrits dans la Section 4.3. Les combinaisons et les identifiants employés dans la suite du discours sont présentés dans le Tableau 6.2. Nous avons ajouté aux variantes *floues* de la PSO une variante originale, de notre conception, qui consiste à faire varier le paramètre de localisation de la QPSO.

Identifiant	Algorithme de base	Inertie	Bornes Vitesse	Facteur Social	Localisation
PSO	PSO				N/A
FI-PSO	PSO	■			N/A
FIV-PSO	PSO	■	■		N/A
FS-PSO	PSO			■	N/A
FIVS-PSO	PSO	■	■	■	N/A
QPSO	QPSO	N/A	N/A	N/A	
FL-QPSO	QPSO	N/A	N/A	N/A	■

TABLE 6.2 – **Présence des contrôleurs flous dans les différentes variantes.** La présence d'un contrôleur flou est indiquée par les cases pleines. Les contrôles inapplicables sont également indiqués par la mention N/A.

Les solutions obtenues par ces variantes sur notre ensemble de fonctions-tests sont reprises dans le Tableau 6.3. Les solutions sont en réalité les valeurs moyennes des solutions obtenues par les algorithmes sur 100 exécutions indépendantes, au moyen d'un ensemble de 100 échantillonnages de départ et de leurs 100 graines aléatoires associées.

Deux faits notables sont à retenir de ce tableau, d'une part nous avons un résultat très intéressant de la part de la FL-QPSO qui l'emporte sur 8 des 12 fonctions, sans jamais être la pire méthode pour une fonction. Au contraire, la FIVS-PSO n'est la meilleure que sur Griewank et se retrouve inefficace sur la plupart des fonctions. Ce phénomène est une manifestation de l'émergence d'un comportement nuisible suite à l'interaction de la PSO et du contrôleur de paramètre social, alors que ce contrôleur donne de bons résultats dans l'implémentation de Nobile [23]. Cette implémentation présente entre autre une topologie totalement connectée, changeant le rôle que doit remplir ce contrôleur.

Parvenir à organiser un plus grand nombre de contrôleurs est un défi étant donné que leur logique et leur paramétrisation dépendent de l'algorithme, de la topologie et des autres contrô-

	PSO	FI-PSO	FIV-PSO	FS-PSO	FIVS-PSO	QPSO	FL-QPSO
Ackley	5.41E+00	5.09E+00	5.31E+00	1.72E+01	2.95E+00	9.63E+00	**2.19E+00**
Rastrigin	4.25E+02	**3.80E+01**	1.14E+02	4.46E+02	3.95E+02	3.92E+02	9.56E+01
Rosenbrock	1.33E+02	5.59E+01	5.11E+01	1.85E+03	5.46E+01	2.30E+02	**4.95E+01**
Alpine1	7.48E-01	1.57E+00	1.21E+00	2.05E+01	4.09E+00	**1.34E-06**	1.54E-01
Alpine2	2.63E+22	2.63E+22	2.63E+22	2.63E+22	2.63E+22	2.63E+22	**2.63E+22**
Schwefel	1.55E+04	1.58E+04	1.58E+04	1.56E+04	1.55E+04	1.52E+04	**1.39E+04**
Griewank	3.10E+00	1.49E+00	3.86E-01	1.75E+02	**3.56E-02**	1.92E+01	1.11E+00
Qing	3.55E+08	7.39E+05	**6.02E+04**	2.68E+10	5.64E+09	6.38E+08	2.74E+05
Michalewicz	3.49E+01	3.43E+01	3.49E+01	3.55E+01	3.59E+01	3.40E+01	**3.09E+01**
Xinsheyang1	7.43E+02	2.26E-01	5.31E-01	6.96E+10	4.08E+12	9.17E+01	**1.35E-02**
Xinsheyang2	5.23E-12	7.34E-12	2.90E-15	4.01E-13	2.88E-17	2.06E-12	**6.04E-18**
Stiblinskitank	6.78E+02	2.23E+02	2.12E+02	8.95E+02	9.91E+02	5.65E+02	**1.44E+02**

TABLE 6.3 – **Solutions moyennes obtenues par les algorithmes à paramètres flous.** Les valeurs présentées sont une moyenne sur 100 exécutions indépendantes des algorithmes. Les valeurs en rouge sont les pires pour la fonction, les valeurs en gras sont les meilleures pour la fonction.

leurs. Une piste envisagée pour maitriser cette complexité est l'optimisation systématique de la structure même des contrôleurs et surtout de leurs paramètres **a**. L'avantage de l'approche par logique floue est d'être très lisible pour un humain, ce qui est parfaitement adapté à cette première approche exploratoire. Cependant, l'emploi d'une fonction de contrôle telle qu'un réseau de neurone serait une voie plus prometteuse afin d'effectuer une optimisation systématique.

En conclusion, nous noterons que ces contrôles de paramètres et de topologies inspirés de la littérature sont capables d'impacter positivement la performance de la PSO mais qu'il faut également être prudent vis à vis de l'interaction des différents contrôleurs qui pourrait dégrader l'algorithme. En ce domaine, la FL-QPSO nous montre à quel point un contrôle simple sur un seul paramètre améliore déjà la qualité de la solution.

6.3 Hybridation de deux variantes au moyen de la logique floue

Dans cette section, nous allons présenter les résultats obtenus par un algorithme mélangeant les deux variantes de PSO dont nous avons discuté : PSO quantique et PSO classique. Cette variante de PSO est notée HPSO. La HPSO statique possède un taux de mutation de 0.5. Les différentes variantes testées sont présentées dans le Tableau 6.4 et leurs solutions obtenues sont reprises dans le tableau 6.5.

Identifiant	Algorithme	Inertie	Bornes Vitesse	Topologie	Hybridation	Localisation
FL-QPSO	QPSO				N/A	
FIV-HPSO	HPSO					
FIVH-HPSO	HPSO					
FIVHT-HPSO	HPSO					

TABLE 6.4 – **Présence des contrôleurs flous dans les différentes variantes.** La présence d'un contrôleur flou est indiquée par les cases pleines. Les contrôles inapplicables sont également indiqués par la mention N/A.

Nous constatons dans le Tableau 6.5 que l'hybridation des algorithmes classiques et quantiques a principalement apporté une augmentation des performances sur Ackley, Alpine1, Griewank et Qing, de quatre à six ordres de grandeurs. Les trois premières fonctions sont caractérisées par un profil global unimodal et un profil local fortement multimodal. L'hybridation a permi aux particules de s'échapper des minimas locaux trompeurs caractéristiques de ces fonctions. Les deux variantes à hybridation dynamique se distinguent par leur capacité à obtenir les résultats

	FQPSO	FIV-PSO	FIV-HPSO	FIVH-HPSO	FIVHT-HPSO
Ackley	2.19E+00	5.31E+00	1.24E-03	2.74E-04	**2.06E-04**
Rastrigin	9.56E+01	1.14E+02	4.12E+02	3.96E+02	**5.57E+01**
Rosenbrock	4.95E+01	5.11E+01	4.61E+01	4.59E+01	**4.59E+01**
Alpine1	1.54E-01	1.21E+00	3.36E-07	2.32E-07	**1.95E-07**
Alpine2	2.63E+22	2.63E+22	2.63E+22	2.63E+22	**2.62E+22**
Schwefel	1.39E+04	1.58E+04	1.54E+04	1.55E+04	**1.23E+04**
Griewank	1.11E+00	3.86E-01	1.41E-02	**3.46E-07**	1.03E-02
Qing	2.74E+05	6.02E+04	7.01E-02	1.36E-02	**1.15E-02**
Michalewicz	**3.09E+01**	3.49E+01	3.45E+01	3.46E+01	3.25E+01
Xinsheyang1	1.35E-02	5.31E-01	2.51E-02	2.65E-02	**1.30E-02**
Xinsheyang2	**6.04E-18**	2.90E-15	6.83E-13	5.75E-13	2.54E-13
Stiblinskitank	**1.44E+02**	2.12E+02	3.16E+02	2.05E+02	1.58E+02

TABLE 6.5 – **Solutions moyennes obtenues par les algorithmes à paramètres flous.** Les valeurs présentées sont une moyenne sur 100 exécutions indépendantes des algorithmes. Les valeurs en rouge sont les pires pour la fonction, les valeurs en gras sont les meilleures pour la fonction.

les plus robustes au sein de l'ensemble des cas tests.

Nous noterons enfin que les performances sur des fonctions telles que Schwefel ne sont pas significativement améliorées par nos variantes. La difficulté de Schwefel réside dans la disposition de ses minimas locaux aux coins du domaine de recherche, la fonction demande une très grande capacité d'exploration qui pourrait être amenée par un mécanisme similaire aux mutations de l'algorithme génétique. Par exemple via l'implémentation d'un contrôleur Catfish PSO [24] qui réinitialiserait les particules les moins performantes de l'essaim à des positions aléatoires condensées aux bords du domaine.

6.4 Comportement et défauts du contrôleur

Dans cette section, nous nous penchons de plus près sur l'influence des contrôleurs en logique floue sur le processus de recherche. Pour support, nous prenons le profil de convergence de la fonction Rastrigin auquel nous avons accolé l'évolution de la variable δ. Pour rappel, cette variable caractérise la distance d'une particule à son meilleur ami. Nous remarquons ainsi à la Figure 6.1 que les profiles de δ et d'aptitude moyenne sur la fonction Rastrigin sont liés pour les deux variantes considérées. La variante HPSO est simplement l'algorithme hybridé sans aucun contrôleur flou et l'algorithme FIVHT-HPSO est notre dernière variante possédant de multiples contrôleurs.

Nous n'avons représenté que la valeur moyenne de δ sur l'essaim et pas celle de ϕ car cette dernière présente une variance trop importante sur les 40 particules et une distribution anormale. Cependant, on peut interpréter la dérivée du profil de convergence à la Figure 6.1 en associant les valeurs SAME ou WORSE à la stagnation de l'aptitude et vice-versa.

Une première caractéristique intrigante sur ce graphe est la différence entre les valeurs initiales de δ pour les deux variantes. Ce phénomène s'explique par la connexion plus faible de la topologie floue au début du processus d'optimisation, favorisant l'exploration libre de l'espace de paramètres.

Nous pouvons ensuite nous attarder sur les points particuliers de la courbe δ. Nous remarquons que δ commence par diminuer, signe que les particules tendent à se confondre avec leur guide topologique, en effet, le profil d'aptitude commence à stagner et en réaction, nous voyons que δ se met à augmenter. Cet effet est obtenu grâce aux mécanismes d'exploration tels que le contrôle inertiel qui ventilent l'essaim. Du point 2 au point 3, nous observons encore une fois

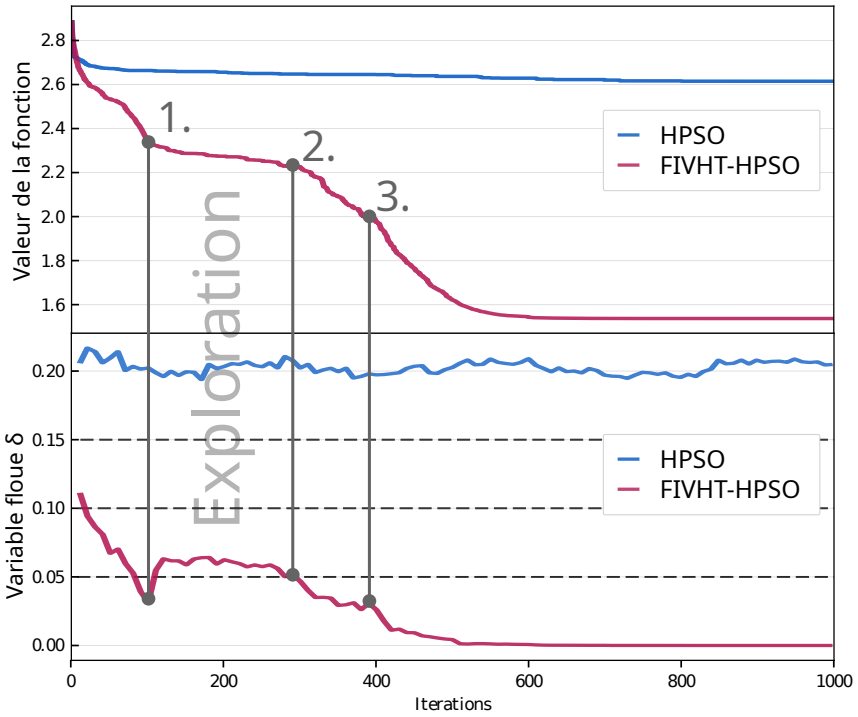


FIGURE 6.1 – **Profil de convergence de l'essaim hybride avec et sans contrôleurs flous.** La variable floue δ moyenne sur l'essaim est présentée aux côtés du profil de convergence. La valeur moyenne sur l'essaim de ϕ n'est pas lisible sur un graphe mais il est possible de s'en donner une impression à partir de la pente du profil de convergence.

une progression de l'essaim.

Après le point 3, nous observons le défaut majeur de notre implémentation. En effet, les évaluation suivantes ne conduisent plus à une amélioration de l'essaim, malgré les mécanismes permettant de favoriser l'exploration. Ceci est dû au fait que ces mécanismes sont basés uniquement sur la vitesse minimale et l'inertie. Ces comportements sont suffisants pour éviter une immobilisation de l'essaim mais insuffisants pour relancer l'exploration. Pour y parvenir, nous pourrions envisager l'implémentation au moyen d'un contrôleur flou de Catfish PSO, méthode que nous avons déjà citée plus haut pour son potentiel à améliorer l'exploration de solutions originales.

6.5 Comparaison à l'algorithme génétique

Dans les sections précédentes, nous avons présenté l'apport de la logique floue à l'algorithme PSO. Dans cette section, nous allons comparer cette heuristique PSO floue à l'algorithme génétique de Minamo, développé par CENAERO, noté Minamo-GA.

Les paramètres employés sont repris dans le Tableau 6.6

De manière générale, l'algorithme PSO de base possède une performance moindre que cet algorithme, à la fois en terme de qualité de solution (Tableau 6.7) qu'en terme d'efficacité (Figures 6.2). Notre algorithme PSO proactif parvient à inverser cette tendance sur la majorité des cas que nous avons explorés.

Il reste cependant des cas tels que Schwefel ou Xin-She Yang qui sont encore un défi pour la PSO. Schwefel de par son profile trompeur et Xin-She Yang à cause de sa forme locale aléatoire perturbant nos capteurs flous. En effet, la variation aléatoire de l'aptitude dans ce cas conduit à un capteur ϕ disruptif. Une solution à ce problème serait d'implémenter un moyenne déroulante qui jouerait le rôle de filtre passe-bas pour les capteurs flous.

Paramètre	Valeur
Population	
Size	40
Mutation Operator	
Probabilité de mutation	0.01
Portée de mutation	0.1
Précision de mutation	32.0
Opérateur de sélection	
Ordre du tournoi	2
Extrapolation	0.5
Directed Rate	0.25

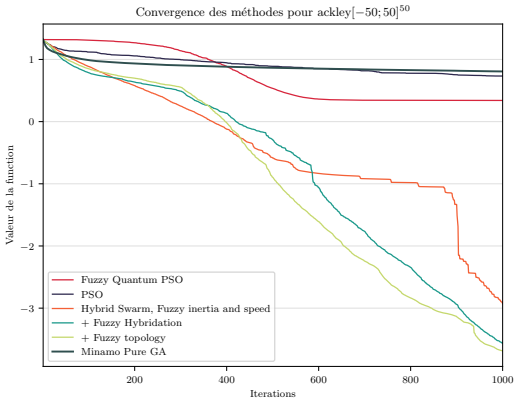
TABLE 6.6 – Paramètres employés pour l'algorithme génétique MINAMO.

Quant au défi pausé par Schwefel, nous envisageons deux pistes. La première, déjà présentée dans ce chapitre consiste à introduire une mécanique d'exploration plus pressante telle qu'une mutation. La seconde piste consisterait à hybrider la PSO avec une autre heuristique, par exemple l'algorithme génétique.

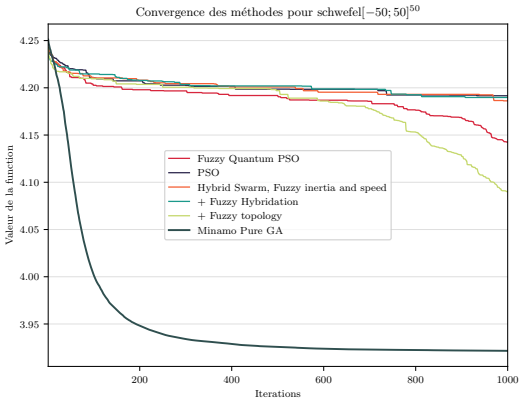
Enfin, comme indiqué dans la section précédente, nous remarquons que le budget d'évaluations n'est pas toujours exploité au mieux sur toutes les fonctions. Nous noterons cependant, toujours à la Figure 6.2 que la forme du profil de convergence montre bien que les alternatives floues de la PSO présentent bien une phase d'exploration accrue durant les premières itérations, conduisant à une grande valeur de l'aptitude minimale, suivies de plongeons exploratoires. Ces comportements plus variés, moins monotones, illustrent l'émergence d'une intelligence d'essaim plus complexe, capable de faire émerger des comportements exploratoires ou d'exploration à des moments, nous le souhaitons, judicieux.

	FQPSO	FIV-PSO	FIV-HPSO	FIVH-HPSO	FIVHT-HPSO	Minamo-GA
Ackley	1.95E+00	5.30E+00	1.27E-01	2.34E-02	**2.08E-02**	6.41E+00
Rastrigin	8.91E+01	4.23E+02	4.05E+02	3.89E+02	**5.47E+01**	5.74E+01
Rosenbrock	5.04E+01	1.37E+02	4.64E+01	4.62E+01	**4.57E+01**	4.99E+01
Alpine1	2.05E-01	6.77E-01	1.25E-06	1.17E-06	**5.45E-08**	1.93E-01
Alpine2	2.63E+22	2.63E+22	2.63E+22	2.63E+22	**2.63E+22**	2.63E+22
Schwefel	1.42E+04	1.57E+04	1.55E+04	1.56E+04	1.35E+04	**8.35E+03**
Griewank	1.08E+00	3.40E+00	7.08E-03	**4.03E-03**	6.02E-03	8.60E-01
Qing	2.10E+05	3.61E+08	7.34E-02	**1.16E-02**	4.18E-02	2.68E+04
Michalewicz	3.06E+01	3.47E+01	3.47E+01	3.49E+01	3.40E+01	**8.63E+00**
Xinsheyang1	**1.40E-02**	1.51E+03	2.55E-02	2.38E-02	1.91E-02	9.90E-02
Xinsheyang2	6.41E-18	1.23E-11	1.30E-12	1.29E-12	1.15E-12	**8.28E-20**
Stiblinskitank	**1.43E+02**	6.38E+02	2.02E+02	2.37E+02	1.60E+02	2.64E+02

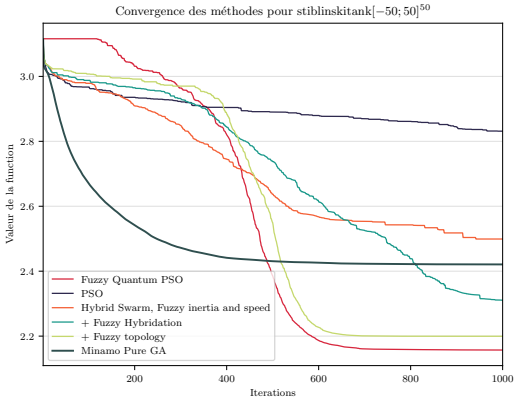
TABLE 6.7 – **Comparaison des performances à l'algorithme génétique.**



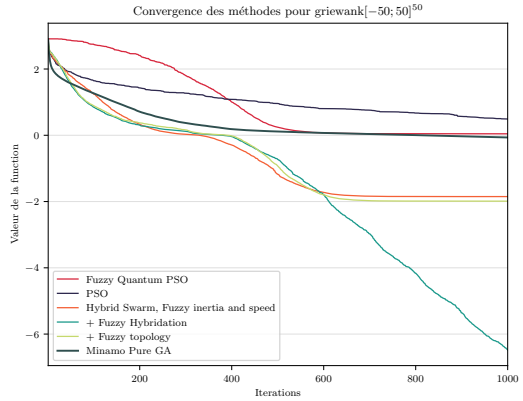
(a) Ackley.



(b) Schwefel.



(c) Styblinsky-Tank.



(d) Griewank.

FIGURE 6.2 – **Évolution de l'aptitude sur différents cas pour quatre fonctions** La valeur de l'aptitude est représentée par le logarithme en base 10 de l'aptitude moyenne sur 100 exécutions indépendantes de l'algorithme.

CHAPITRE 7

Conclusions et perspectives

Au terme de ce manuscrit il est désormais temps de se rappeler les principaux résultats de ce travail. Dans celui-ci, nous nous sommes lancé le défi de contourner le problème du *No Free Lunch Theorem*. Notre fil rouge fut l'implémentation d'un moteur de réflexion au sein de chaque individu de l'algorithme PSO.

Tout d'abord, nous avons mis au point un cadre de travail pour le contrôle de paramètres et d'hybridation dynamique pour un algorithme heuristique, en particulier, la PSO. Nous sommes parvenus à deux constats majeurs : un essaim au comportement plus complexe peut être plus efficace sur un ensemble de problèmes. Cependant, cette complexité nécessite une grande précaution dans la définition des règles logiques et des métaparamètres appliqués.

Au niveau des performances de l'algorithme hybride, nous sommes parvenus à intégrer des avantages de la Quantum PSO, l'hybridation apportant une réelle plus-value quand dynamiquement intégrée.

Nous pouvons à présent parler des perspectives ouvertes par ce travail. Nous avons cité dans le manuscrit la possibilité d'implémenter un contrôleur de Catfish PSO, ceci afin d'apporter une solution relançant l'exploration afin d'employer au mieux le budget d'évaluations alloué. Une autre piste est l'emploi d'une heuristique alternative à la PSO pour l'hybridation. Cette implémentation serait certes plus complexe mais nous avons montré que l'algorithme génétique reste plus adapté à certains problèmes. Et pour aller plus loin, nous pouvons nous ouvrir à une hybridation multiple avec des algorithmes totalement différents.

Nous avons également lancé une piste pour des développements futurs : l'emploi d'un réseau neuronal pour établir le lien entre les sondes et les paramètres à contrôler. Le réseau serait ainsi entrainé sur un ensemble de problèmes de référence afin que les particules apprennent un processus d'optimisation robuste.

Pour finir, dans l'optique d'intégrer cet algorithme dans une application concrète, nous comptons l'exploitation de métamodèles et l'intégration de la gestion des contraintes au cœur des perspectives liées à ce travail.

Annexes

ANNEXE A

Description des fonctions

A.1 Fonction de Ackley

L'expression de cette fonction est donnée par

$$f(\mathbf{x}) = -a.e^{-b\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}} - e^{\frac{1}{n}\sum_{i=1}^{n}\cos(cx_i)} + a + e,$$

où n représente la dimension du domaine de recherche. Les bornes du domaine admissible sont telles que

$$\mathbf{x} \in [-32.775, 32.775]^n.$$

Le profil de la fonction est représenté pour $n = 2$ à la Figure A.1.

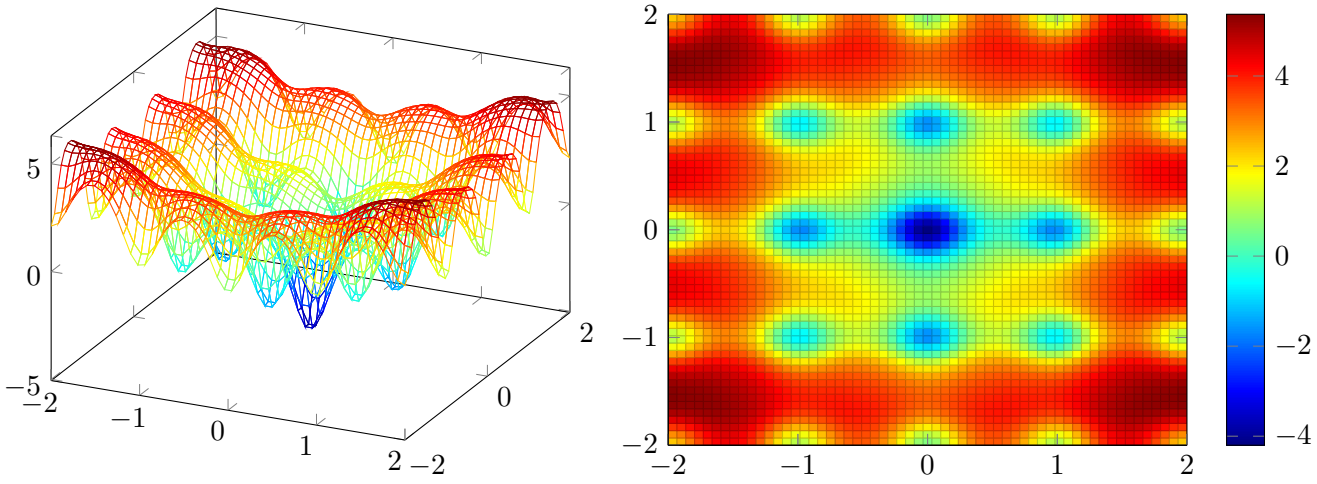


FIGURE A.1 – Profil de la fonction ackley

A.2 Première fonction d'Alpine

L'expression de cette fonction est donnée par

$$f(\mathbf{x}) = \sum i = 1^n |x_i \sin(x_i) + 0.1x_i|$$

où n représente la dimension du domaine de recherche. Les bornes du domaine admissible sont telles que

$$\mathbf{x} \in [0.0, 10.0]^n.$$

Le profil de la fonction est représenté pour $n = 2$ à la Figure A.2.

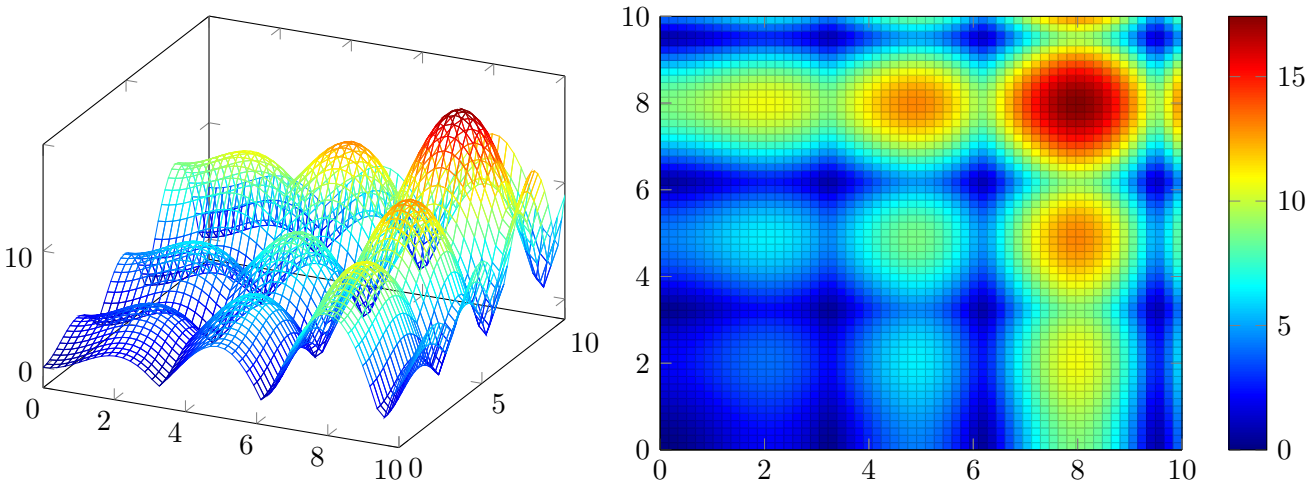


FIGURE A.2 – Profil de la fonction alpine1

A.3 Seconde fonction d'Alpine

L'expression de cette fonction est donnée par

$$f(\mathbf{x}) = \prod_{i=1}^{n} \sqrt{x_i} \sin(x_i)$$

où n représente la dimension du domaine de recherche. Les bornes du domaine admissible sont telles que

$$\mathbf{x} \in [0.0, 10.0]^n.$$

Le profil de la fonction est représenté pour $n = 2$ à la Figure A.3.

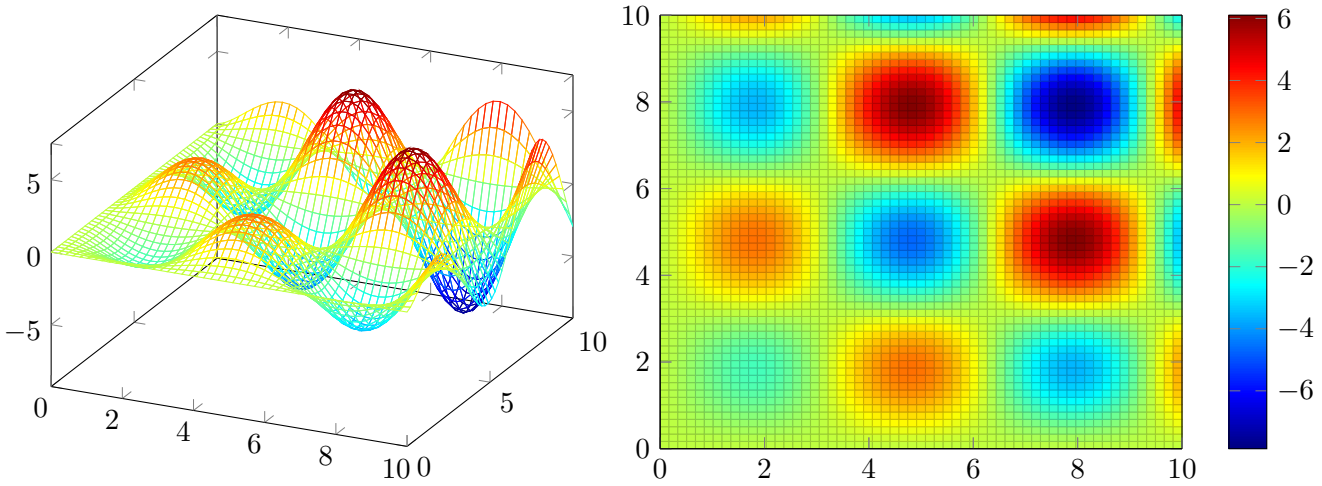


FIGURE A.3 – Profil de la fonction alpine2

A.4 Fonction de Griewank

L'expression de cette fonction est donnée par

$$f(\mathbf{x}) = 1 + \sum_{i=1}^{n} \frac{x_i^2}{4000} - \prod_{i=1}^{n} \cos(\frac{x_i}{\sqrt{i}})$$

où n représente la dimension du domaine de recherche. Les bornes du domaine admissible sont telles que

$$\mathbf{x} \in [-512.0, 512.0]^n.$$

Le profil de la fonction est représenté pour $n = 2$ à la Figure A.4.

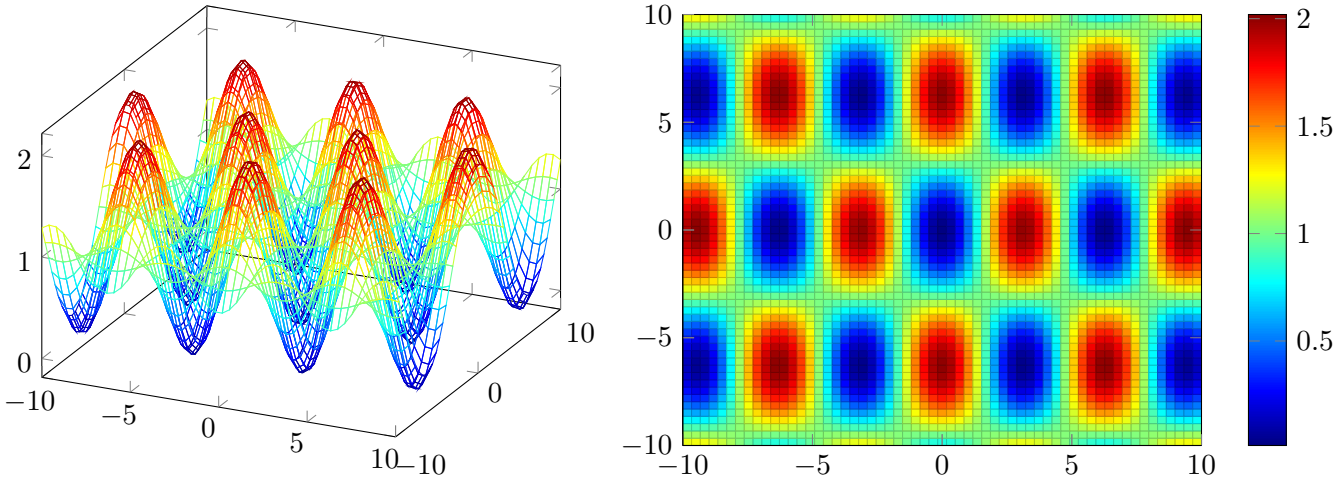


FIGURE A.4 – Profil de la fonction griewanl

A.5 Fonction de Qing

L'expression de cette fonction est donnée par

$$f(\mathbf{x}) = \sum_{i=1}^{n} (x^2 - i)^2$$

où n représente la dimension du domaine de recherche. Les bornes du domaine admissible sont telles que

$$\mathbf{x} \in [-500.0, 500.0]^n.$$

Le profil de la fonction est représenté pour $n = 2$ à la Figure A.5.

A.6 Fonction de Rastrigin

L'expression de cette fonction est donnée par

$$f(\mathbf{x}) = 10n + \sum_{i=1}^{n} (x_i^2 - 10\cos(2\pi x_i))$$

où n représente la dimension du domaine de recherche. Les bornes du domaine admissible sont telles que

$$\mathbf{x} \in [-5.12, 5.12]^n.$$

Le profil de la fonction est représenté pour $n = 2$ à la Figure A.6.

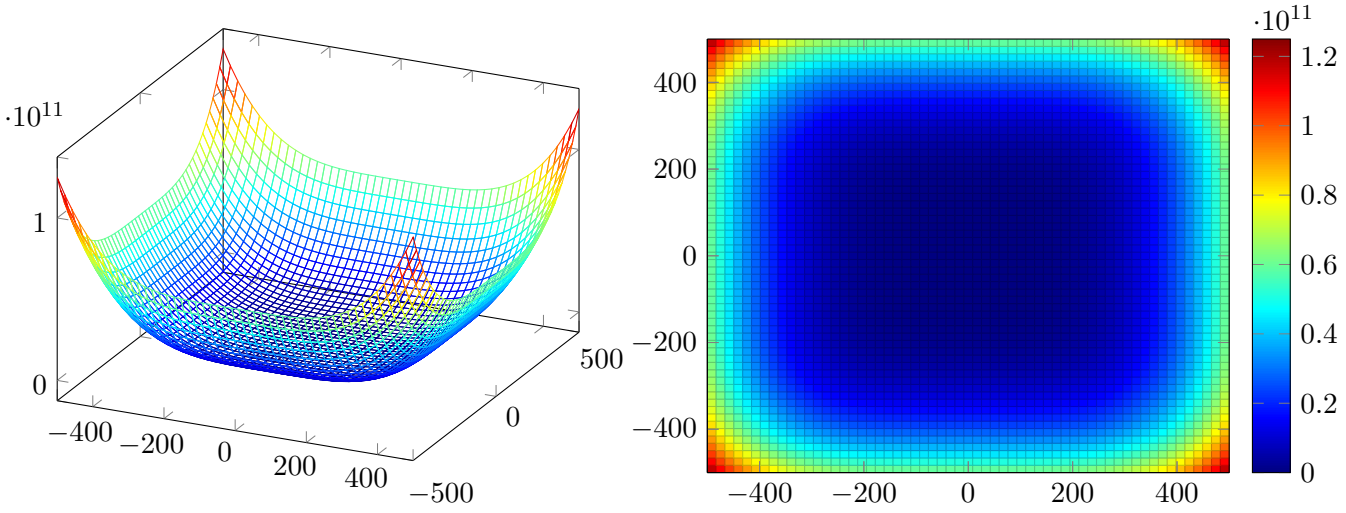


FIGURE A.5 – Profil de la fonction qing

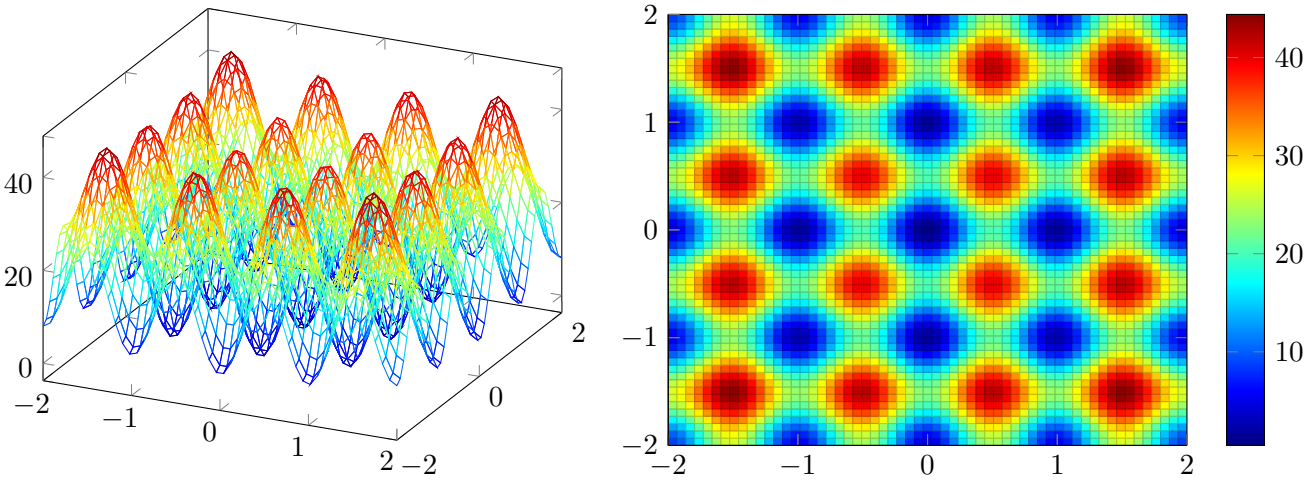


FIGURE A.6 – Profil de la fonction rast

A.7 Vallée de Rosenbrock

L'expression de cette fonction est donnée par

$$f(\mathbf{x}) = \sum_{i=1}^{n} [b(x_{i+1} - x_i^2)^2 + (a - x_i)^2]$$

où n représente la dimension du domaine de recherche. Les bornes du domaine admissible sont telles que

$$\mathbf{x} \in [-2.048, 2.048]^n.$$

Le profil de la fonction est représenté pour $n = 2$ à la Figure A.7.

A.8 Fonction de Michalewicz

L'expression de cette fonction est donnée par

$$f(\mathbf{x}) = -\sum_{i=1}^{n} \sin x_i \sin^{2n} \frac{i x_i^2}{\pi}$$

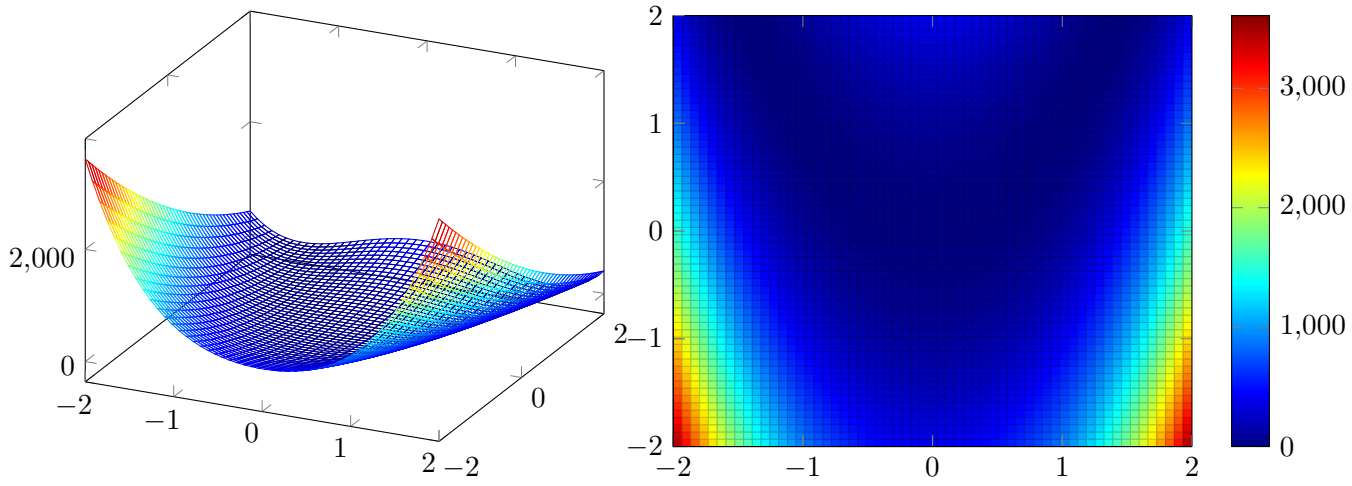


FIGURE A.7 – Profil de la fonction rosen

où n représente la dimension du domaine de recherche. Les bornes du domaine admissible sont telles que

$$\mathbf{x} \in [0.0, \pi]^n.$$

Le profil de la fonction est représenté pour $n = 2$ à la Figure A.8.

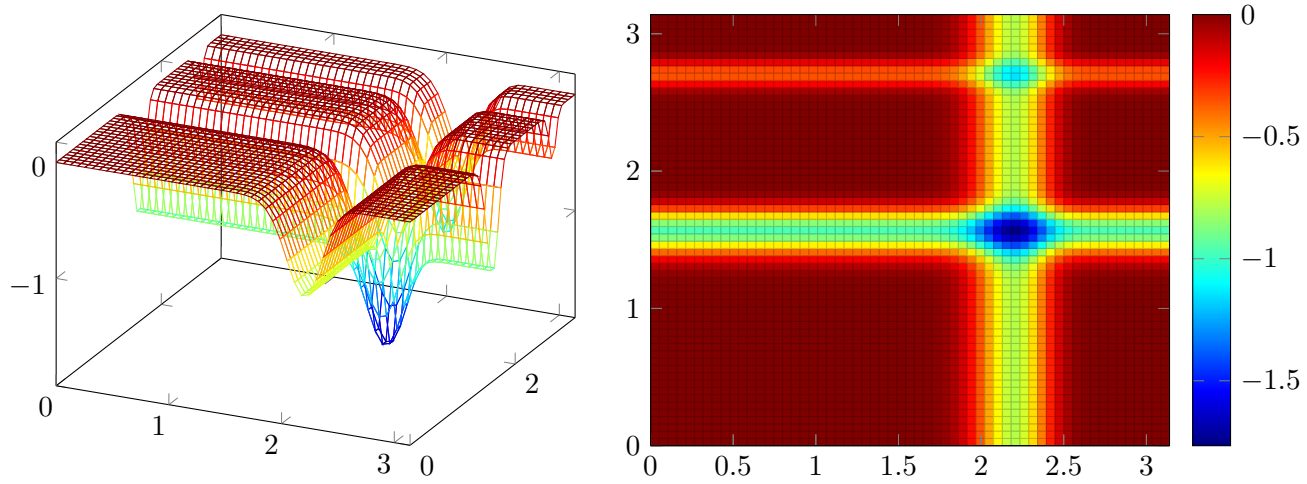


FIGURE A.8 – Profil de la fonction micha

A.9 Fonction de Styblinski-Tank

L'expression de cette fonction est donnée par

$$f(\mathbf{x}) = f(x_1, ..., x_n) = \frac{1}{2}\sum_{i=1}^{n}(x_i^4 - 16x_i^2 + 5x_i)$$

où n représente la dimension du domaine de recherche. Les bornes du domaine admissible sont telles que

$$\mathbf{x} \in [-5.0, 5.0]^n.$$

A.10 7ème fonction de Schwefel

L'expression de cette fonction est donnée par

$$f(\mathbf{x}) = -\sum_{i=1}^{n} x_i \sin\left(\sqrt{|x_i|}\right)$$

où n représente la dimension du domaine de recherche. Les bornes du domaine admissible sont telles que

$$\mathbf{x} \in [-500.0, 500.0]^n.$$

Le profil de la fonction est représenté pour $n = 2$ à la Figure A.9.

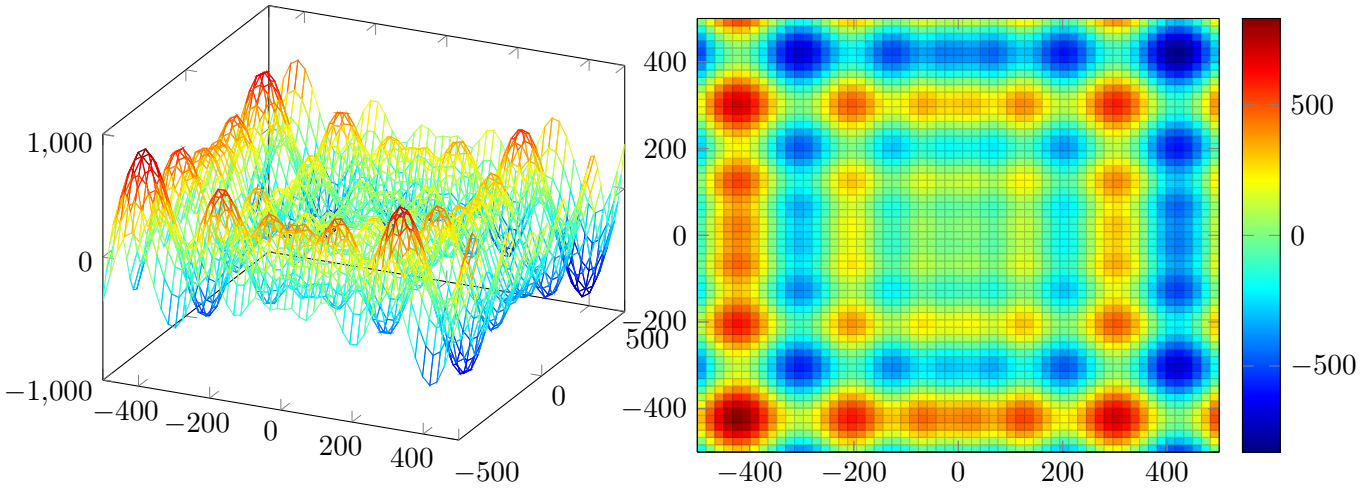


FIGURE A.9 – Profil de la fonction stib

A.11 Première fonction de Xin-She Yang

L'expression de cette fonction est donnée par

$$f(\mathbf{x}) = f(x_1, ..., x_n) = \sum_{i=1}^{n} \epsilon_i |x_i|^i$$

où n représente la dimension du domaine de recherche. Les bornes du domaine admissible sont telles que

$$\mathbf{x} \in [-5.0, 5.0]^n.$$

Le profil de la fonction est représenté pour $n = 2$ à la Figure A.10.

A.12 Seconde fonction de Xin-She Yang

L'expression de cette fonction est donnée par

$$f(\mathbf{x}) = f(x_1, ..., x_n) = (\sum_{i=1}^{n} |x_i|) e^{-\sum_{i=1}^{n} sin(x_i^2)}$$

où n représente la dimension du domaine de recherche. Les bornes du domaine admissible sont telles que

$$\mathbf{x} \in [-6.2832, 6.2832]^n.$$

Le profil de la fonction est représenté pour $n = 2$ à la Figure A.11.

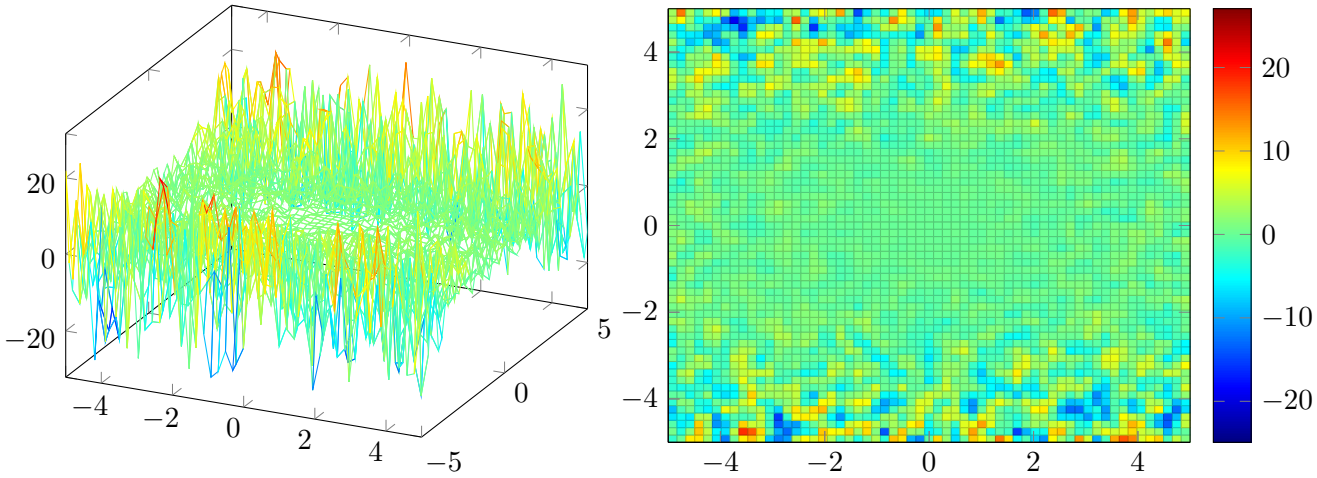


FIGURE A.10 – Profil de la fonction xin1

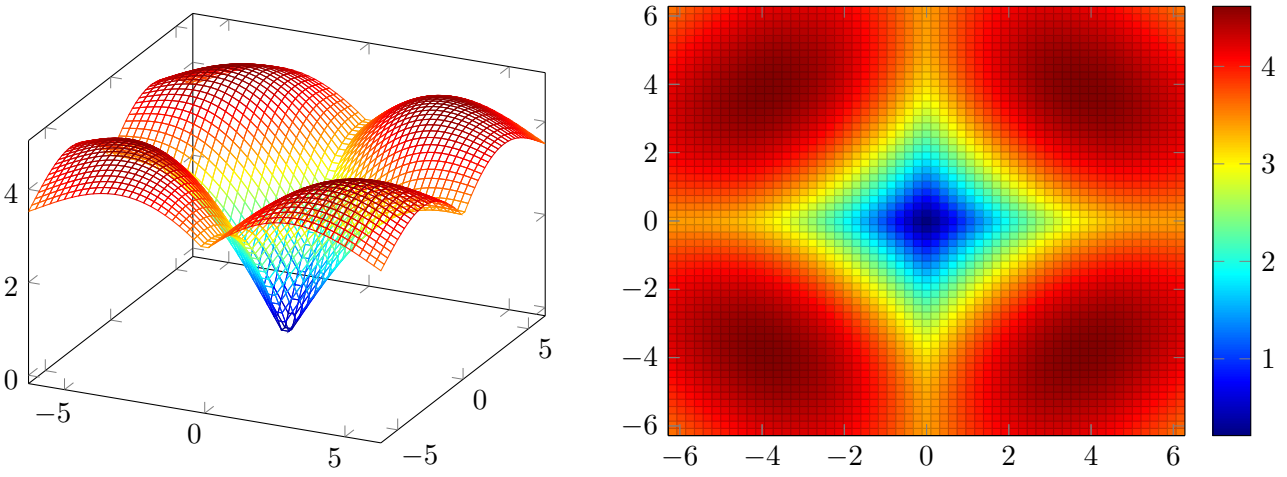


FIGURE A.11 – Profil de la fonction xin2

Bibliographie

[1] C. Blum and A. Roli, "Metaheuristics in combinatorial optimization : Overview and conceptual comparison," *ACM computing surveys (CSUR)*, vol. 35, no. 3, pp. 268–308, 2003.

[2] R. Eberhart and J. Kennedy, "A new optimizer using particle swarm theory," in *MHS'95. Proceedings of the Sixth International Symposium on Micro Machine and Human Science*, pp. 39–43, Ieee, 1995.

[3] J. Sun, B. Feng, and W. Xu, "Particle swarm optimization with particles having quantum behavior," in *Proceedings of the 2004 congress on evolutionary computation (IEEE Cat. No. 04TH8753)*, vol. 1, pp. 325–331, IEEE, 2004.

[4] M. Clerc, "Standard particle swarm optimisation from 2006 to 2011," 2012.

[5] D. H. Wolpert, W. G. Macready, *et al.*, "No free lunch theorems for optimization," *IEEE transactions on evolutionary computation*, vol. 1, no. 1, pp. 67–82, 1997.

[6] Y. Shi and R. C. Eberhart, "Fuzzy adaptive particle swarm optimization," in *Proceedings of the 2001 Congress on Evolutionary Computation (IEEE Cat. No. 01TH8546)*, vol. 1, pp. 101–106, IEEE, 2001.

[7] M. Zambrano-Bigiarini, M. Clerc, and R. Rojas, "Standard particle swarm optimisation 2011 at CEC-2013 : A baseline for future PSO improvements," pp. 2337–2344, IEEE, 2013.

[8] J. Xin, G. Chen, and Y. Hai, "A particle swarm optimizer with multi-stage linearly-decreasing inertia weight," in *2009 International Joint Conference on Computational Sciences and Optimization*, vol. 1, pp. 505–508, IEEE, 2009.

[9] H. H. Hoos and T. Stützle, *Stochastic local search : Foundations and applications*. Elsevier, 2004.

[10] M. Dorigo and T. Stützle, "Ant colony optimization : overview and recent advances," in *Handbook of metaheuristics*, pp. 311–351, Springer, 2019.

[11] J. B. Escario, J. F. Jimenez, and J. M. Giron-Sierra, "Ant colony extended : experiments on the travelling salesman problem," *Expert Systems with Applications*, vol. 42, no. 1, pp. 390–410, 2015.

[12] C. W. Reynolds, "Flocks, herds and schools : A distributed behavioral model," in *ACM SIGGRAPH computer graphics*, vol. 21, pp. 25–34, ACM, 1987.

[13] M. D. McKay, R. J. Beckman, and W. J. Conover, "Comparison of three methods for selecting values of input variables in the analysis of output from a computer code," *Technometrics*, vol. 21, no. 2, pp. 239–245, 1979.

[14] Y. Shi and R. Eberhart, "A modified particle swarm optimizer," in *IEEE world congress on computational intelligence (Cat. No. 98TH8360)*, pp. 69–73, IEEE, 1998.

[15] Y. Shi, "Brain storm optimization algorithm," in *International conference in swarm intelligence*, pp. 303–309, Springer, 2011.

[16] J. Kennedy, "Small worlds and mega-minds : effects of neighborhood topology on particle swarm performance," in *Proceedings of the 1999 Congress on Evolutionary Computation-CEC99 (Cat. No. 99TH8406)*, vol. 3, pp. 1931–1938, IEEE, 1999.

[17] B. C. Roy Nicolas, "Implémentation de l'optimisation par essaims de particules dans la platerforme minamo, cenaero," 2018.

[18] W. Van Leekwijck and E. E. Kerre, "Defuzzification : criteria and classification," *Fuzzy sets and systems*, vol. 108, no. 2, pp. 159–178, 1999.

[19] M. Sugeno, *Industrial Applications of Fuzzy Control*. Elsevier Amsterdam, 1985.

[20] M. S. Nobile, G. Pasi, P. Cazzaniga, D. Besozzi, R. Colombo, and G. Mauri, "Proactive particles in swarm optimization : a self-tuning algorithm based on fuzzy logic," in *2015 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, pp. 1–8, IEEE, 2015.

[21] J. C. Bansal, P. Singh, M. Saraswat, A. Verma, S. S. Jadon, and A. Abraham, "Inertia weight strategies in particle swarm optimization," in *2011 Third world congress on nature and biologically inspired computing*, pp. 633–640, IEEE, 2011.

[22] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns : Elements of Reusable Object-oriented Software*. Boston, MA, USA : Addison-Wesley Longman Publishing Co., Inc., 1995.

[23] M. S. Nobile, P. Cazzaniga, D. Besozzi, R. Colombo, G. Mauri, and G. Pasi, "Fuzzy self-tuning pso : A settings-free algorithm for global optimization," *Swarm and evolutionary computation*, vol. 39, pp. 70–85, 2018.

[24] L.-Y. Chuang, S.-W. Tsai, and C.-H. Yang, "Catfish particle swarm optimization," in *2008 IEEE Swarm Intelligence Symposium*, pp. 1–5, IEEE, 2008.